ENCYCLOPEDIA FOR THE TRS-80*

A library of useful information for your TRS-80

Business
Education
Games
Graphics
Hardware
Home Applications
Interface
Tutorial
Utility

VOLUME 1

ENCYCLOPEDIA for the TRS-80*

VOLUME 1

wayne GREEN inc.
PETERBOROUGH NH 03458

*Trademark of Tandy Corp.

FIRST EDITION
FIRST PRINTING APRIL 1981

Printed in the United States of America

Edited by Chris Crocker, Nan McCarthy
and Susan Philbrick
Technical Assistance: Dennis Bathory Kitsz
and Dennis Thurlow
Design by Clare McCarthy
Illustrations by Howard Happ
Typeset by Karen Stewart

FOREWORD

The Biggest Difference

There are lots of arguments about which computer is the best. The answer to this question lies not in which hardware is best. That is really irrelevant, when you understand the field. The major value of any computer lies in the software and the information available for it. Hence this encyclopedia.

The TRS-80 is by no means the best computer on the market as far as its hardware is concerned, but with the support of *80 Microcomputing* magazine and this encyclopedia series, you have an almost unlimited source of information on how to use your computer—and of programs. With this information source the TRS-80 is by far the most valuable computer system ever built. No other computer, at any price, has anything approaching this amount of user information and programs available.

Most encyclopedias try to freeze everything at one time and are thus able to divide the material up alphabetically. This is a new kind of encyclopedia —a living one—with each new volume keeping you up to date on the very latest information on using your computer and the newest of programs.

Your computer can be a fantastic teaching device, a simulator, a way to play all sorts of fascinating games, a business aid, a scientific instrument, a control unit for machinery. . . . It is one of the most flexible gadgets ever invented. All of these applications are possible *if* you have the information and the programs. This encyclopedia will give you these.

To get the best use of your TRS-80, don't miss a single volume of the *Encyclopedia for the TRS-80.*

WAYNE GREEN
Publisher

CONTENTS

Please note: Before typing in any listing in this book, see Appendix A.

CONTENTS

CONTENTS

APPENDICES

BUSINESS

Down the Road

After the Goldrush

Business Forms:
The Invoice

How Much Interest?
—The Rule of 78

Down the Road

by Bill Vick

As a regional sales manager with a major consumer products company, I am constantly challenged with finding new and creative ways of filling the product pipeline between my company and the consumer. This pipeline, or channel of distribution, consists of department stores.

The way a store projects its future sales and determines the stock needed to support those sales is a basic financial planning concept, applicable to large or small businesses.

Our company now furnishes a stock and sales plan designed to maximize profits, as well as a profitability and return on investment analysis. These reports are generated on my TRS-80, and this article covers the planning process and its application.

The Retail Season

Retailers periodically go through a planning process forecasting sales for a given period of time, called a season. Normally a season runs six months and is referred to as a "spring" or "fall" season. Whether the sales forecast is done with linear regression, exponential smoothing, S.O.T.P. (seat of the pants), or any other means, the end result is that merchandise must be purchased in advance of planned sales. The control and management of the flow of inventory through a retailer determines whether or not a profit is made.

```
WHAT YEAR IS THE PLAN FOR (YY)          ?80
WHAT IS THE CHAIN'S NAME                ?ANY STORE
WHAT IS THE BRANCH/DOOR'S NAME          ?MAIN ST.
WHICH SEASON WILL THIS PLAN COVER
(FALL OR SPRING)                        ?SPRING

ENTER THE 79 SALES FOLLOWED BY A COMMA, THEN THE 80 SALES
  79/80 SALES FOR FEB? 11.1,12.2
  79/80 SALES FOR MAR? 10.9,13.1
  79/80 SALES FOR APR? 11.5,10.7
  79/80 SALES FOR MAY? 12.4,15.2
  79/80 SALES FOR JUN? 9.2,9.9
  79/80 SALES FOR JLY? 11.6,11.9

PLANNED SEASON TURN? 1.39
IF YOU HAVE AN ACTUAL BOM ENTER IT.
IF YOU DO NOT PRESS ENTER ?—
```

Example 1

Plans are normally drafted for both seasons and the fiscal year. A turnover ratio of stock to sales is predetermined, based on the needs of both the retailer and the vendors involved. In a simplified way, turnover is the number of times an average inventory will replace itself as it is sold.

As an example, if over a six month period, or season, sales were $10,000, with an average inventory of $8,000, the turnover ratio would be 1.25 for the season. The turnover ratio is critical. Too fast a turnover means in all probability sales were lost as a result of being out of stock, while too slow a turnover can eat into profit and increase your overhead. Generally, the faster the turnover, the greater the profit.

Two Seasons

In my business, we work on two seasons a year, which go from February through July and August through January. The average inventory turnover is 3.0 per year, or 1.4 turns in the spring and 1.6 in the fall.

The following program determines monthly stock levels and planned purchases needed to maintain stock levels. A sales history for the prior period is shown, along with the B.O.M. (beginning of month) inventory, the E.O.M. (end of month) inventory, and the allowed monthly receipts or O.T.R. (open to receive).

—1980 STOCK/SALES PLAN FOR ANY STORE—MAIN ST.

		FEB	MAR	APR	MAY	JUN	JLY	TOT
79 SALES		11.1	10.9	11.5	12.4	9.2	11.6	66.7
80 SALES		12.2	13.1	10.7	15.2	9.9	11.9	73.0
BOM		52.6						
EOM		53.5	51.1	55.6	50.3	52.3	52.5	
OTR		13.1	10.7	15.2	9.9	11.9	12.2	
TURN 1.39	AVG INV 52.5			TOTAL SALES 73.0				+9%

Example 2

Additionally, the average inventory carried on hand and the percent of increase or decrease planned are also shown. It is an interactive program designed to be used by people not versed in either computerese or retailing.

The advantages of this type of planning are many. Not only can the retailer's stock and sales plans be done with speed and accuracy, but a number of "what if" situations can quickly be accomplished to determine viable alternatives.

Program Listing

```
100 CLEAR 1000:
    CLS
110 REM  *   STOCK AND SALES PLANNING
120 REM  *   BILL VICK   1/30/80
130 REM  *
140 PRINT TAB(20)"STOCK AND SALES PLANNING"
150 PRINT TAB(25)"BY BILL VICK"
160 PRINT TAB(22)"ALL RIGHTS RESERVED"
170 FOR T = 1 TO 500:
     NEXT T
180 DIM M(13),M$(13),EOM(13),Z$(2),OTB(13),S(13)
190 RESTORE :
    CLS
200 INPUT "WHAT YEAR IS THE PLAN FOR? (YY) ";YEAR$
210 YEAR$ = RIGHT$(YEAR$,2):
    YEAR = VAL(YEAR$):
    LYEAR = YEAR - 1
220 INPUT "WHAT IS THE CHAIN'S NAME? ";CHAIN$
230 INPUT "WHAT IS THE BRANCH/DOOR'S NAME? ";BRANCH$
240 FOR M = 00 TO 12:
     READ M$(M):
     NEXT :
    REM  * READ IN MONTHS
250 DATA JAN,FEB,MAR,APR,MAY,JUN,JLY,AUG,SEP,OCT,NOV,DEC,JAN
260 PRINT "WHICH SEASON WILL THIS PLAN COVER (FALL OR SPRING) ";
270 GOSUB 820:
    REM  * INKEY ROUTINE
280 IF KB$ = "S"
     THEN
      PRINT "SPRING" :
     ELSE
      PRINT "FALL"
290 PRINT
300 IF KB$ = "S"
     THEN
      SB = 1:
      SE = 6:
      S = 1:
      GOTO 330
310 IF KB$ = "F"
     THEN
      SB = 7:
      SE = 12:
      S = 2:
      GOTO 330
320 IF KB$ < > "S" AND KB$ < > "F"
     THEN
      CLS :
      GOTO 260
330 PRINT "ENTER THE ";LYEAR;" SALES FOLLOWED BY A COMMA, THEN THE "
    ;YEAR;" SALES"
340 FOR T = SB TO SE
350  PRINT LYEAR;"/";YEAR;"  SALES FOR  ";M$(T);
360  INPUT S(T),M(T)
370  ST = ST + S(T):
     REM  * TOTAL SALES THIS YEAR
380  TS = TS + M(T):
     REM  * TOTAL SALES LAST YEAR
390  NEXT
400 AV = ( INT(TS - ST) / ST) * 100:
    REM  * PERCENT INCREASE IN SALES
410 PRINT :
    INPUT "PLANNED SEASON TURN ";TURN
420 PRINT "IF YOU HAVE AN ACTUAL BOM ENTER IT"
430 INPUT "IF YOU DO NOT, PRESS ENTER ";BOM
440 AS = TS / 6:
    REM  * AVERAGE MONTHS SALES = TOTAL SALES/SIX MONTHS
450 AI = TS / TURN:
    REM  * AVERAGE INVENTORY = TOTAL SALES/TURN
460 IF BOM < = 0
     THEN
      470 :
     ELSE
      480
```

Program continued

```
470 IF M(SB) > AS
     THEN
      BOM = (M(SB) - AS) + AI :
     ELSE
      BOM = (AS - M(SB)) + AI
480 Z$(1) = "          FEB     MAR      APR     MAY    JUN     JLY   TOT"
490 Z$(2) = "          AUG     SEP      OCT     NOV    DEC     JAN     TOT
    "
500 N$ = "EOM      ###.#   ###.#   ###.#   ###.#   ###.#   ###.#"
510 BY$ = "OTR      ###.#   ###.#   ###.#   ###.#   ###.#   ###.#"
520 OS$ = "BOM      ###.#"
530 R$ = "##SALES ###.#   ###.#   ###.#   ###.#   ###.#   ###.#   ###.#"
540 L$ = "##SALES ###.#   ###.#   ###.#   ###.#   ###.#   ###.#   ###.#"
550 Q$ = "TURN ##.##    AVE INV ###.#    TOT SALES   ###.#    +####%"
560 FOR T = SB TO SE - 1:
    REM  * COMPUTES EOM INVENTORY
570  EOM(T) = (M(T + 1) - AS) + AI
580  EOM = EOM + EOM(T):
    REM  * TOTAL EOM FOR SEASON
590  NEXT
600 CN = (AI * 7) - (BOM + EOM):
    REM  * COMPUTES ENDING PERIOD INV.
610 EOM(SE) = CN
620 YEAR = YEAR + 1900
630 CLS :
    PRINT YEAR;" STOCK/SALES PLAN FOR   ";CHAIN$;" - ";BRANCH$
640 YEAR = YEAR - 1900
650 PRINT
660 PRINT Z$(S)
670 PRINT
680 PRINT USING L$; LYEAR, S(SB), S(SB + 1), S(SB + 2), S(SB
    + 3), S(SB + 4), S(SB + 5), ST
690 PRINT USING R$; YEAR, M(SB), M(SB + 1), M(SB + 2), M(SB
    + 3), M(SB + 4), M(SB + 5), TS
700 PRINT :
    PRINT USING OS$; BOM
710 PRINT USING N$; EOM(SB), EOM(SB + 1), EOM(SB + 2), EOM(SB
    + 3), EOM(SB + 4), EOM(SB + 5)
720 PRINT :
    PRINT USING BY$; M(SB + 1), M(SB + 2), M(SB + 3), M(SB + 4), M(S
    B + 5), M(SB)
730 PRINT
740 AI = (EOM + EOM(SE) + BOM) / 7:
    REM  * AVERAGE INVENTORY
750 PRINT USING Q$; TURN, AI, TS, AV
760 PRINT :
    GOSUB 840
770 PRINT "DO YOU WANT TO PRINT THIS? (Y/N) "
780 GOSUB 820:
    REM  * INKEY ROUTINE
790 IF LEFT$(KB$,1) = "Y"
     THEN
      GOSUB 850:
      GOTO 620:
      REM  * IF YES LPRINTS
800 IF KB$ = "N" RUN 180:
    REM  * IF NO RERUNS PROGRAM
810 IF KB$ < > "N" AND KB$ < > "Y"
     THEN
      770
820 KB$ = INKEY$:
    IF KB$ = ""
     THEN
      820
830 KB = VAL(KB$):
    RETURN
840 POKE 16414,88:
    POKE 16415,4:
    RETURN :
    REM  * LPRINT TO PRINT
850 POKE 16414,141:
    POKE 16415,5:
RETURN :
REM  * PRINT TO LPRINT
```

After the Goldrush

by Jerry Frost

Many of you will say, "But I don't have bags of silver or gold chains." You may surprise yourself when you find that Uncle Walter's Masonic ring or Grandpa's pocket watch has more than sentimental value. A close examination of silver coins left in your bureau, baby cups, and cuff links will tell if they are sterling or 14K or 18K gold.

The accompanying program will store your inventory of gold and silver and produce an up-to-the minute account of these holdings compared to the daily spot prices in any of the world's precious metal markets—New York, London, Paris, Zurich, Hong Kong.

The market analysis section of the program will tell, at a glance, the percentage of gain or loss on your holdings, as gold and silver continue to climb.

Tipping the Scales

The first thing to do is to determine, as accurately as possible, the actual pure gold or silver content of that class ring or sterling teapot. Obviously, weighing them with a bathroom scale won't do unless, of course, you possess a hundred pounds or so of these precious metals. The best solution is to use a jeweler's scale.

Since most of us don't have one, you'll want to visit your local jeweler and, for a fee, have your cache weighed. If you have a postage scale at the office, you'll get a fairly accurate measurement in avoirdupois ounces.

Precious metals are currently weighed in troy ounces in the United States and Canada as a standard of measurement.

Simply multiply avoirdupois ounces by .9114583 to obtain the equivalent troy weight. For example, weigh a sterling silver spoon on a standard scale and observe a weight of 1.5 avoirdupois ounces. Multiplying 1.5 by .9114583 gives you a troy ounce weight of 1.367 ounces.

This is only a gross weight, not the actual pure silver content. All sterling silver has non-precious metals added to it as hardeners. Fineness, therefore, is defined as being that part of the metal alloy containing pure gold or silver. Sterling silver has 925 parts silver in 1000 parts alloy. You must now find the pure silver weight of the sterling spoon: Multiply .925 by the gross weight of 1.367 troy ounces. This yields 1.264 troy ounces of *pure* silver, expressed in what's called "1000 fine."

Pure gold is considered to be 24 karats. The relation of fineness to karats is also proportional. A 14K gold ring, for example, contains 583.3 parts gold in 1000 parts of alloy. An 18K ring would contain 750 parts gold in 1000 parts of alloy. Weigh the ring or any other gold item, then convert it to troy ounces and multiply by its fineness. Table 1 shows the conversion of karats to fineness.

24	karats =	1000 fine	20	karats =	833.3 fine
23	karats =	958.3 fine	18	karats =	750. fine
22	karats =	916.6 fine	16	karats =	666.7 fine
21.6	karats =	900.0 fine	14	karats =	583.3 fine
21	karats =	875.0 fine	1	karat =	041.7 fine

Table 1

A warning: Do not weigh different karat items together; combine all 14K jewelry, all 18K, etc., and weigh them separately.

A magnifying glass will help you see the karat stamp on jewelry. Beware of any gold item stamped *G.P.* or *G.F.* This means the piece of jewelry is gold plated or filled. It is not a solid gold alloy. So, don't waste your time weighing these items.

Fineness

Both United States and foreign gold and silver coins contain various amounts of fineness. Table 2 lists the most common intrinsic domestic and foreign gold coins with their pure troy ounce content. Multiply this weight by the number of coins you have.

U.S.	$20	gold piece	.9675
	$10	gold piece	.4838
	$5	gold piece	.2419
	$2.50	gold piece	.1209
	$1.00	gold piece	.0483

Table 2

U.S. silver coins minted through 1964 contain 90 percent silver. Clad fifty-cent pieces minted from 1965 through 1970 contain 40 percent silver. Coin dealers and precious metal buyers consider that a $1000 face value bag of circulated United States coins minted through 1964 contains about 720 troy ounces of silver, while a $1000 face value bag of circulated Kennedy silver clad half dollars minted from 1965 through 1970 contains about 295 troy ounces.

All United States coins (other than some proof sets minted for collectors) minted after 1970 are nothing more than copper clad coins with no silver content whatsoever!

Foreign coins are another source of silver. Some countries even stamp the purity and weight right on the coin. If you aren't sure, a trip to a local coin dealer or library will tell if there is treasure in that hoard. An excellent coin catalog, *Standard Catalog of World Coins,* is published by Krause Publishers, Iola, Wisconsin. You'll find a reference to your coin and its silver content in this catalog.

Inventory Program

Once the groundwork has been laid and all of your gold and silver holdings accurately measured, converted to troy ounces, and their fineness determined, you're ready to enter inventory data statements in a program.

The program lists the following information: description, quantity, pure troy weight (in ounces), and original cost (or close estimate). Refer to Table 3(a) for examples and proper format. Make sure that the last statement in the inventory of precious metals data line always terminates with END.

```
20000 REM * INVENTORY OF PRECIOUS METALS *
20010 DATA #14K JEWELRY, 1, 1.75, 250
20020 DATA *STERLING SILVER, 1, 120, 680
20030 DATA *STERLING KNIVES, 8, 1.20, 75
20040 DATA END
```

Table 3(a)

```
30000 REM * CLOSING DATES & SPOT PRICES *
30010 DATA #01/21/80, 850
30020 DATA *01/21/80, 50
30030 DATA END
```

Table 3(b)

The computer will have to determine whether your data is of gold or silver. To do this, precede the description and spot price dates with the marker # for gold and * for silver. Therefore lines 20010 and 30010 refer to gold, while lines 20020, 20030, and 30020 refer to silver. The marker will be stripped for all CRT displays and printouts.

Referring to line 20030, notice that if you include sterling knives, they are listed separately from other silverware. This is because knife handles are usually hollow and filled with wax. The blade is often made of stainless steel. A good rule of thumb is to weigh the knife and take two-fifths of the total weight as sterling content.

The quantity number 1 in line 20010 means that you gathered your 14 karat gold jewelry as a group, weighed it, and came up with 1.75 total troy ounces. The eight knives in line 20030 were weighed separately, giving a weight of 1.20 troy ounces. The program takes the quantity eight and multiplies it by 1.20 for a total weight of 9.6 troy ounces. This is for the convenience of those who wish to list their gold and silver items separately.

Lines 30010 and 30020 keep tab on the daily market closing price. You can consult the business sections of most newspapers to obtain this data. Line 30010 shows, for example, that on January 21, 1980, gold closed at $850 an ounce, while line 30020 shows that on the same day, silver closed at $50 an ounce.

You can enter new data daily, weekly, or monthly to keep up with the fluctuating bullion market, as compared to the latest spot metals price. Always terminate the last closing dates and spot prices line with END.

The program needs no explanation. The input commands are self-prompting. If you require hard copy (recommended), just change PRINTs to LPRINTs. Better yet, if you're using a disk system with NEWDOS (also

1 troy ounce	= 31.1033 grams
1 troy ounce	= 480 grains
1 troy ounce	= 20 pennyweight (DWT)
12 troy ounces	= 1 pound troy
14.5833 troy ounces	= 1 pound avoirdupois
0.9114 troy ounces	= 1 ounce avoirdupois
32.15 troy ounces	= 1 kilogram
1 gram	= 5.3 karats (Roman)
1 gram	= 15.432 grains
1 gram	= 0.643 pennyweight (DWT)
1.5552 grams	= 1 pennyweight (DWT)
1,000 grams	= 1 kilogram
28.3495 grams	= 1 ounce avoirdupois
24 grains	= 1 pennyweight (DWT)
5,760 grains	= 1 pound troy
15,432 grains	= 1 kilogram
437.5 grains	= 1 ounce avoirdupois
7,000 grains	= 1 pound avoirdupois
1 grain	= 0.0648 grams
240 pennyweight (DWT)	= 1 pound troy
643.01 pennyweight (DWT)	= 1 kilogram
18.2291 pennyweight (DWT)	= 1 ounce avoirdupois
291.666 pennyweight (DWT)	= 1 pound avoirdupois
1 kilogram	= 2.68 pounds troy
1 kilogram	= 35.274 ounces avoirdupois
1 kilogram	= 2.2046 pounds avoirdupois

Table 4

recommended), simply hit the JKL keys simultaneously and you'll get a hard copy of the screen displays. If you require larger arrays, increase at line 800.

After creating your data statements, selecting menu item 4 will automatically re-SAVE the program (METAL/BAS) and data to disk. A sequential or random file method could be used, but I feel the method of re-SAVING is adequate for this data management without increasing the size and complexity of the program. Cassette users must change the SAVE "METALS/BAS" to CSAVE "METAL" in line 2200. It is good practice to keep a separate copy of your program in case of I/O errors.

Other Metals

You can incorporate other precious metals—platinum, for example—in the program. You may also want to keep track of the price of copper. That lowly penny in your pocket may someday be worth more for its intrinsic value than for its monetary value!

To include these or other metals in the program, first create additional menu lines between lines 1200 and 1500. Then edit lines 2900 and 4900, inserting new markers denoting the new metals. Any uppercase symbols such as % and ! will do. You'll have to add IF statements between lines 1900 and 2200. Edit line 2300. Be sure to precede all data lines with the new marker(s).

After the program is run, the first display produces an itemized inventory of your precious metal holdings. The MKT. VALUE (market value) column tells, at a glance, its current value. The COST column refers to your original investment. The CHANGE column gives the percentage of difference between the current market value and the initial cost. The automatic scrolling feature of the program allows you to pause between displays.

The next display contains the current total dollar value of your investment, compared to the original value. These holdings are represented in pure 1000 fine troy ounces.

The final display is an up-to-the-minute market analysis showing past closing dates and closing spot prices and the percentage of change from the current spot price of the metal in question.

This analysis allows you to keep up with the volatile activity in the precious metals exchange and to record its history. The automatic scrolling pauses between these displays.

Another addition to the program will help determine the pure troy ounce content of your holdings. Although troy ounces are used, you may refer to Table 4 and convert most common weights to troy ounces. United States silver coins don't have to be weighed because the program will do it for you. Enter the face value and its percentage (90 percent or 40 percent) of silver.

Now delete the example data lines, 20010 through 30090, and add your own. Run the program and see how "loded" you are.

Program Listing

```
 700 CLEAR 1000
 800 DIM M$(50),Q(50),F(50)
 900 CLS
1000 PRINT :
     PRINT :
     PRINT :
     PRINT TAB(25)"* MENU *"
1100 PRINT :
     PRINT
1200 PRINT TAB(15)"1 - GOLD MARKET ANALYSIS"
1300 PRINT TAB(15)"2 - SILVER MARKET ANALYSIS"
1400 PRINT TAB(15)"3 - TROY OUNCE WEIGHT CALCULATION"
1500 PRINT TAB(15)"4 - WRITE NEW DATA STATEMENTS TO DISK"
1600 N$ = INKEY$:
     IF N$ = "" GOTO 1600
1700 N = VAL(N$)
1800 CLS
1900 IF N = 1
      THEN
       GS$ = "GOLD"
2000 IF N = 2
      THEN
       GS$ = "SILVER"
2100 IF N = 3
      THEN
       7200
2200 IF N = 4
      THEN
       PRINT @590,"";:
       INPUT "HIT <ENTER> TO SAVE NEW DATA";X$:
       PRINT @580,"NOW RE-WRITING PROGRAM AND ADDING NEW DATA TO DISK
       ":
       SAVE "METALS/BAS":
       RUN
2300 IF N < 1 OR N > 3
      THEN
       900
2400 PRINT TAB(25)GS$;" ANALYSIS"
2500 PRINT TAB(20) STRING$(23,131)
2600 PRINT :
     PRINT
2700 PRINT "<ENTER> CURRENT SPOT  * ";GS$;:
     INPUT " * PRICE PER TROY OUNCE ";P
2800 PRINT
2900 IF N = 1
      THEN
       R$ = "*":
      ELSE
       IF N = 2
        THEN
         R$ = "#":
         REM  *SET DATA MARKER *
3000 INPUT "<ENTER> TODAY'S DATE (MM/DD/YY) ";D$
3100 FOR X = 1 TO 50
3200  READ M$(X)
3300  IF M$(X) = "END"
       THEN
        X = X - 1:
        Z = X:
        GOTO 3700
3400  READ Q(X),F(X),C(X)
3500  IF LEFT$(M$(X),1) = R$
       THEN
        X = X - 1:
        REM  * READ DATA MARKER *
3600  NEXT X
3700 FOR X = 1 TO Z:
```

Program continued

```
      MV(X) = P * F(X) * Q(X):
      MV = MV + MV(X):
      C = C + C(X):
      Q = Q + Q(X):
      F = F + F(X) * Q(X)
3800  NEXT X
3900 CLS
4000 GOSUB 6300:
     GOTO 4100
4100 FOR X = 1 TO Z
4200  PRINT USING "###";Q(X);:
      PRINT TAB(6) RIGHT$(M$(X), LEN(M$(X)) - 1);:
      PRINT TAB(31) USING "##,###.##";MV(X);:
      PRINT TAB(42) USING "##,###.##";C(X);:
      PRINT TAB(54) USING "+####.#";((MV(X) - C(X)) / C(X) * 100);:
      PRINT " %"
4300  ZZ = ZZ + 1:
      IF ZZ = 10
       THEN
        ZZ = 0:
        PRINT STRING$(63,45):
        GOSUB 6200:
        IF X = Z GOTO 4700:
       ELSE
        GOSUB 6300
4400  NEXT
4500 GOSUB 6200
4600 PRINT STRING$(8,32):
     PRINT STRING$(63,45)
4700 PRINT TAB(8)"CURRENT MARKET VALUE = $";:
     PRINT USING "##,###.##";MV
4800 PRINT TAB(9)"ORIGINAL INVESTMENT = $";:
     PRINT USING "##,###.##";C
4900 IF N = 1
      THEN
       R$ = "*":
      ELSE
       IF N = 2
        THEN
         R$ = "#":
         REM  * SET DATA MARKER *
5000 PRINT :
     PRINT TAB(10)"REPRESENTING ";:
     PRINT USING "###.##";F;:
     PRINT " TROY OUNCES OF 1000 FINE ";GS$
5100 PRINT STRING$(63,45)
5200 GOSUB 6200:
     GOSUB 7000:
     ZZ = 0:
     GOTO 5300
5300 FOR X = 1 TO 50:
      READ D$(X)
5400  IF D$(X) = "END"
       THEN
        Z = X:
        GOTO 5800
5500  READ SP(X)
5600  IF LEFT$(D$(X),1) = R$
       THEN
        X = X - 1:
        REM  * READ DATA MARKER *
5700  NEXT X
5800 Z = Z - 1:
     FOR X = 1 TO Z:
      PRINT RIGHT$(D$(X), LEN(D$(X)) - 1),:
      PRINT USING "#,###.##";SP(X);:
      PRINT , USING "+###.##";((P - SP(X)) / SP(X) * 100);:
      PRINT " %"
5900  ZZ = ZZ + 1:
      IF ZZ = 10
       THEN
```

```
        PRINT STRING$(63,45):
        ZZ = 0:
        GOSUB 6200:
        IF X = Z GOTO 6100
6000  NEXT X
6100 PRINT @980,"PRESS <ENTER> RETURN TO MENU";:
     LINE INPUT A$:
     RUN
6200 PRINT @980,"PRESS <ENTER> TO CONTINUE";:
     LINE INPUT A$:
     CLS :
     RETURN
6300 PRINT D$; TAB(20)GS$" PORTFOLIO"; TAB(46)"SPOT = $";:
     PRINT USING "#,###.#";P
6400 PRINT TAB(15) STRING$(25,61)
6500 PRINT
6600 PRINT STRING$(63,45)
6700 PRINT "QTY"; TAB(10)"DESCRIPTION"; TAB(32)"MKT. VALUE";
     TAB(46)"COST"; TAB(55)"CHANGE"
6800 PRINT STRING$(63,45)
6900 RETURN
7000 PRINT D$; TAB(15)GS$" MARKET ANALYSIS"; TAB(46)"SPOT = $";:
     PRINT USING "#,###.##";P:
     PRINT TAB(15) STRING$(23,61):
     PRINT :
     PRINT STRING$(63,45):
     PRINT "CLOSE DATE"; TAB(19)"SPOT"; TAB(34)"CHANGE TO DATE":
     PRINT STRING$(63,45)
7100 RETURN
7200 :
     '         * GOLD & SILVER TROY OUNCE WEIGHT *
7300 CLS
7400 PRINT TAB(25)" * MENU *"
7500 PRINT :
     PRINT
7600 PRINT TAB(15)"1 - GOLD CALCULATION"
7700 PRINT TAB(15)"2 - SILVER CALCULATION"
7800 N$ = INKEY$:
     IF N$ = "" GOTO 7800
7900 CLS
8000 N = VAL(N$)
8100 IF N = 2 GOTO 10000
8200 CLS
8300 PRINT TAB(15)"GOLD CONVERSION TABLE"
8400 PRINT TAB(15) STRING$(21,45)
8500 PRINT :
     PRINT
8600 INPUT "<ENTER> KARAT WEIGHT OF GOLD ITEM ";K
8700 K = .041666667 * K
8800 PRINT :
     PRINT
8900 INPUT "<ENTER> WEIGHT SYSTEM:  1 - AVOIRDUPOIS    2 - TROY ";AT
9000 IF AT < 1 OR AT > 2 GOTO 8900
9100 IF AT = 1AT = .9114583:
      ELSE
       AT = 1
9200 PRINT
9300 INPUT "<ENTER> WEIGHT OF GOLD ITEM (OUNCES) ";W
9400 W = W * K * AT
9500 PRINT
9600 PRINT STRING$(46,45)
9700 PRINT "ITEM CONTAINS";:
     PRINT USING "##.###";W;:
     PRINT " TROY OUNCE(S) OF PURE GOLD."
9800 PRINT STRING$(46,45)
9900 GOSUB 6100
10000 PRINT TAB(15)"SILVER CONVERSION TABLE"
10100 PRINT TAB(15) STRING$(23,45)
10200 PRINT
10300 PRINT "<ENTER>  1 - STERLING SILVER    2 - U.S. COINS"
```

Program continued

```
10400 N$ = INKEY$:
      IF N$ = "" GOTO 10400
10500 PRINT @192, STRING$(63,32)
10600 N = VAL(N$)
10700 IF N < 1 OR N > 2 GOTO 10300
10800 IF N = 1N = .925:
      GOTO 12600:
      REM   * .925= STERLING FINENESS *
10900 PRINT
11000 PRINT TAB(10)"1 - 90% PRE-1965 U.S. SILVER COINS"
11100 PRINT
11200 PRINT TAB(10)"2 - 40% 1965-1970 KENNEDY SILVER CLAD HALVES"
11300 X$ = INKEY$:
      IF X$ = "" GOTO 11300
11400 X = VAL(X$)
11500 IF X < 1 OR X > 2 GOTO 11000
11600 IF X = 1X = .72:
      REM  90% SILVER WEIGHT PER $1 FACE VALUE
11700 IF X = 2X = .295:
      REM  40% SILVER WEIGHT PER $1 FACE VALUE
11800 PRINT
11900 INPUT "<ENTER>   FACE VALUE OF U.S. COINS ";FV
12000 FV = FV * X
12100 PRINT
12200 PRINT STRING$(57,45)
12300 PRINT "U.S. COINS CONTAIN ";:
      PRINT USING "#,###.###";FV;:
      PRINT " TROY OUNCE(S) OF PURE SILVER."
12400 PRINT STRING$(57,45)
12500 GOSUB 6100
12600 PRINT
12700 INPUT "<ENTER> WEIGHT SYSTEM:  1-AVOIRDUPOIS    2-TROY ";AT
12800 IF AT < 1 OR AT > 2 GOTO 12700
12900 IF AT = 1 AT = .9114583:
       ELSE
        AT = 1
13000 PRINT :
      PRINT
13100 INPUT "<ENTER> WEIGHT OF STERLING ITEM (OUNCES) ";W
13200 W = W * N * AT
13300 PRINT :
      PRINT
13400 PRINT STRING$(59,45)
13500 PRINT "STERLING ITEM CONTAINS ";:
      PRINT USING "#,###.###";W;:
      PRINT " TROY ONCES OF PURE SILVER."
13600 PRINT STRING$(59,45)
13700 GOSUB 6100
13800 END
20000 REM   *  EXAMPLE INVENTORY DATA LINES  *
20010 DATA #14K JEWELRY,1,1.75,250.00
20020 DATA *STERLING SILVER,1,120,680.00
20030 DATA *STERLING KNIVES,8,1.20,75.00
20040 DATA #$20 U.S. GOLD PIECE,1,.9675,325.00
20050 DATA *$40 FACE 90% U.S. COINS,40,.720,624.00
20060 DATA *$75 FACE 40% U.S. COINS,75,.295,400.00
20070 DATA #18K NECKLACE,1,.475,548.00
20080 DATA END
20090 :
      '
30000 REM  * EXAMPLE CLOSING DATA & SPOT PRICE DATA LINE
30010 DATA #01/21/80,850.00
30020 DATA *01/21/80,50.00
30030 DATA #01/22/80,682.00
30040 DATA #01/30/80,690.00
30050 DATA *01/30/80,34.00
30060 DATA *04/02/80,14.60
30070 DATA #04/02/80,493.00
30080 DATA END
30090 :
      ' END OF LISTING
```

Business Forms: The Invoice

by R. L. Conhaim

Every business, large or small, needs some kind of paperwork to inform its customers of merchandise or services sold. Retail businesses often use cash register receipts or sales slips. But, at the wholesale, services, or manufacturing level, the invoice is the most common sales record. The buyer uses the invoice as an accounts payable record and to price inventory records. The seller uses it as an accounts receivable record and for inventory management.

In the past, the next step up from manual invoice writing was the billing machine. Today, the computer has taken over the invoicing function. In many computer systems, invoicing is interactive with accounts receivable, general ledger, and inventory programs.

The invoice program in this article produces a computer-generated invoice containing the essentials suitable for many businesses. It requires a TRS-80 Level II and a peripheral printer. With slight modification the program can be used with pre-printed invoice forms or, as presented here, it can produce the entire invoice on plain paper. The program also provides for keeping a record of totals on tape or for posting to the business accounting system.

Let's take a closer look at the invoice and the program which generates it. The heading information is part of the program and would not be needed on pre-printed forms. The invoice number for the first invoice of the day is provided by the computer operator. Subsequent invoice numbers are provided in consecutive order by the program. The operator, in response to prompting, enters the date, buyer's purchase order number, terms of the transaction, and the name and address of the buyer. The position on the page of the buyer's name and address has been chosen to be visible in the window of a #10 (business size) window envelope if the invoice is folded in thirds. For any of this information which is to be repeated on the next invoice, the operator merely presses ENTER in response to the appropriate question.

Data for each item sold is supplied by the operator. The item number in the first column is supplied automatically by the computer. Price each is entered by the operator and the total (price times quantity) is calculated by the program. The price each and totals columns are right-justified through the print-using function. Long item descriptions, up to two lines of 34

characters each, can be used. By the use of the LEN function, the computer determines whether a one- or two-line description is needed, so the operator need not count characters. The only limit to the number of items that may be included on one invoice is the size of the sheet.

After the last item is entered, the program prints LAST ITEM, after which

INVOICE

FROM:
GENERAL SUPPLY COMPANY
1234 CENTER STREET
DAYTON, OHIO 45406

INVOICE NUMBER: 1234
DATE: JULY 20 1980
P.O.NO:B6789
TERMS:2%-10 DAYS
NET 30

SUPPLIES AND EQUIPMENT FOR COMMERCIAL & INDUSTRIAL USE

TO:

THE BONKERS CONSTRUCTION CO.
7650 MALDEN LANE
DAYTON OHIO 45406

ITEM	QUAN	DESCRIPTION	PRICE EA.	TOTAL
1	16	#21 D handle shovels	5.67	90.72
2	6	Gross #8 × 1-½" flat head wood screws	2.13	12.78
3	3	Concrete shaker screens size 7	34.90	104.70
4	4	Cragmire #23 wheelbarrows	67.10	268.40
5	6	Stanley #44 nail hammers	6.80	40.80
6	2	Portable tool sheds equipped with master locks	208.80	417.60
		LAST ITEM		

** SUB—TOTAL	935.00
SALES TAX (5.0%)	46.75
SHIPPING CHARGES	8.90
**TOTAL	990.65

YOU MAY DEDUCT 18.70 IF PAID BY JULY 30 1980

INVOICE SUMMARY
NEXT INVOICE NO. 1235

TOTAL MERCHANDISE	935.00
TOTAL POSSIBLE DISCOUNT	18.70
TOTAL SALES TAX	46.75
TOTAL SHIPPING	8.90
GRAND TOTAL	990.65

Example 1. *Sample Invoice*

the subtotal for merchandise is calculated by the computer and printed.

In response to prompting, the operator indicates whether or not the invoice is subject to a prompt-pay discount, sales tax, if any, and shipping charges. The data is calculated automatically and printed. Because buyers typically miscalculate the prompt-pay discount, the program does the calculating for them, indicating the dollar amount that may be deducted by the entered date.

After all items on the invoice have been printed, line 730 signals the printer to advance the paper to the next sheet. When all invoices during a session are completed, the program prints totals for the session of merchandise, possible discounts, sales taxes, shipping costs, and a grand total on a separate sheet. Discounts are not included in the grand total, as not all buyers will take the discount.

The latest totals may then be added to the totals for previous sessions and kept on cassette tape. The cassette of the previous totals is loaded, the new figures added to previous figures, and the result recorded back to the cassette. If desired, a month-to-date summary can then be printed out.

Program Listing

```
10 :
   '  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
20 :
   '  INVOICE PRINT AND TAPE STORAGE PROGRAM
30 :
   '  BY R. L. CONHAIM
50 :
   '  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
60 CLS
70 CLEAR 1000
80 INPUT "ENTER STARTING INVOICE NUMBER";X1
90 INPUT "ENTER DATE";DA$
100 INPUT "ENTER P.O. NUMBER";PO$
110 INPUT "ENTER TERMS";E1$
130 LPRINT TAB(32) CHR$(01);"INVOICE"; CHR$(02):
    LPRINT
150 LPRINT "FROM:"; TAB(46) "INVOICE NUMBER:";X1
160 LPRINT "GENERAL SUPPLY COMPANY"; TAB(46) "DATE: ";DA$
170 LPRINT "1234 CENTER STREET"; TAB(46)"P.O.NO:";PO$
180 LPRINT "DAYTON, OHIO  45406"; TAB(46)"TERMS:";E1$
190 LPRINT
200 LPRINT TAB(11) "SUPPLIES AND EQUIPMENT FOR COMMERCIAL & INDUSTRI
    AL USE"
210 LPRINT STRING$(75,"-")
220 INPUT "NAME";N$
230 INPUT "STREET";A$
240 INPUT "CITY,STATE,ZIP";C$
250 LPRINT "TO:":
    LPRINT
260 LPRINT TAB(15)N$
270 LPRINT TAB(15) A$
280 LPRINT TAB(15) C$
290 LPRINT :
    LPRINT
300 DEFDBL B,J,S,T
310 :
    '  BEGIN ENTRY OF ITEMS
320 LPRINT STRING$(75,"-")
330 LPRINT "ITEM"; TAB(8) "QUAN"; TAB(27) "DESCRIPTION"; TAB(56) "PR
    ICE EA.  TOTAL"
340 LPRINT STRING$(75,"-")
350 LT = LT + 1
360 INPUT "QUANTITY SOLD"; Q
370 INPUT "DESCRIPTION";I$
380 IF LEN(I$) > 34
     THEN
      L1 = 1 :
     ELSE
      L1 = 0
390 INPUT "UNIT PRICE";R
400 M$ = "##,###.##"
410 TP = R * Q:
    ST = ST + TP
420 LPRINT LT; TAB(7)Q;
430 IF L1 = 0 LPRINT TAB(15)I$;:
    GOTO 460;
440 IF L1 = 1
     THEN
      LPRINT TAB(15) LEFT$(I$,34);"-"
450 LPRINT TAB(16) MID$(I$,35);
460 LPRINT TAB(53) USING M$;R;
470 LPRINT TAB(63) USING M$;TP
480 INPUT "ANOTHER ITEM (Y/N)";D$
```

```
490 IF D$ = "Y" GOTO 350:
     ELSE
      500
500 LPRINT TAB(20)"LAST ITEM"
510 LPRINT STRING$(75,"-"):
    LPRINT
520 LPRINT TAB(32)"** SUB-TOTAL"; TAB(63); USING M$;ST
530 INPUT "IS THIS INVOICE SUBJECT TO PROMPT PAY DISCOUNT (Y/N)";D1$
540 IF D1$ = "Y" GOTO 550:
     ELSE
      580
550 INPUT "ENTER DISCOUNT PERCENT (WHOLE NUMBER)";E2
560 INPUT "ENTER DISCOUNT EXPIRATION DATE";DT$
570 Z = (E2 / 100) * ST
580 INPUT "IS THIS INVOICE TAXABLE (Y/N)";D3$
590 IF D3$ = "Y" GOTO 600:
     ELSE
      620
600 OS = .05 * ST
610 LPRINT TAB(32)"SALES TAX (5.0%)"; TAB(63) USING M$;OS
620 INPUT "ENTER SHIPPING CHARGES";F
630 IF F = 0 GOTO 650
640 LPRINT TAB(32)"SHIPPING CHARGES"; TAB(63) USING M$;F
650 T = ST + OS + F
660 LPRINT TAB(37) "**TOTAL"; TAB(63) USING M$;T
670 IF D1$ = "Y" GOTO 680:
     ELSE
      700
680 LPRINT :
    LPRINT ; TAB(8) "YOU MAY DEDUCT "; USING "##.##";Z;
690 LPRINT " IF PAID BY ";DT$
700 B1 = B1 + ST:
    B2 = B2 + Z:
    B3 = B3 + OS:
    B4 = B4 + F:
    B5 = B5 + T
710 X = 1
720 ST = 0:
    Z = 0:
    OS = 0:
    F = 0:
    T = 0:
    X1 = X1 + 1:
    LT = 0
730 LPRINT CHR$(12)
740 INPUT "ANOTHER INVOICE (Y/N)";D4$
750 IF D4$ = "Y" GOTO 100:
     ELSE
      760
760 LPRINT TAB(20)"INVOICE SUMMARY":
    LPRINT
770 LPRINT TAB(15)"NEXT INVOICE NO.";X1:
    LPRINT
780 LPRINT TAB(10)"TOTAL MERCHANDISE"; TAB(35) USING M$;B1
790 LPRINT TAB(10)"TOTAL POSSIBLE DISCOUNT"; TAB(35) USING M$;B2
800 LPRINT TAB(10)"TOTAL SALES TAX"; TAB(35) USING M$;B3
810 LPRINT TAB(10)"TOTAL SHIPPING"; TAB(35) USING M$;B4
820 LPRINT TAB(10)"GRAND TOTAL"; TAB(35) USING M$;B5
830 C1 = 0
840 INPUT "PREPARE CASSETTE TO PLAY. WHEN READY PRESS ENTER";C1
850 IF C1 < > 0 GOTO 830
860 INPUT # - 1,J1,J2,J3,J4,J5
870 J1 = J1 + B1:
    J2 = J2 + B2:
    J3 = J3 + B3:
    J4 = J4 + B4:
    J5 = J5 + B5
880 D1 = 0
```

Program continued

```
 890 INPUT "PREPARE CASSETTE TO RECORD. WHEN READY PRESS ENTER;D1
 900 IF D1 < > 0 GOTO 880
 910 PRINT # - 1,J1,J2,J3,J4,J5
 920 PRINT "REWIND CASSETTE.INVOICES COMPLETED"
 930 INPUT "DO YOU WANT MONTH-TO-DATE SUMMARY (Y/N)";MS$
 940 IF MS$ = "Y" GOTO 2000:
      ELSE
       950
 950 END
2000 INPUT "ENTER TODAY'S DATE";TD$
2050 LPRINT "MONTHLY INVOICE SUMMARY AS OF ";TD$:
     LPRINT
2060 M$ = "##,###.##"
2070 LPRINT "TOTAL MERCHANDISE"; TAB(35) USING M$;J1
2080 LPRINT "TOTAL POSSIBLE DISCOUNT"; TAB(35) USING M$;J2
2090 LPRINT "TOTAL SALES TAX"; TAB(35) USING M$;J3
2100 LPRINT "TOTAL SHIPPING"; TAB(35) USING M$;J4
2110 LPRINT :
     LPRINT TAB(10) "GRAND TOTAL"; TAB(35) USING M$;J5
2120 PRINT "REWIND CASSETTE"
2130 GOTO 950
```

How Much Interest? The Rule of 78

by Charles B. Steele

So you inherit $10,000 from good old Aunt Minnie and decide now is the time to get that auto or furniture loan off your back. The whopping finance charge that was added to the cost of what you bought won't all have to be paid after all. Let's see now, the loan was for 48 months and you've been making monthly payments for two years. If you pay it off, you should get that finance charge cut about in half. Off to the finance company to tell them the good news. They won't have to hound you for those monthly installments any more.

But now for the bad news. When the finance officer hears you want to pay off the loan, he does some calculating and tells you that you get a reduction of only 25.5 percent of the finance charge. You now discover that installments you have been paying were a lot more interest and a lot less principal than you thought. You've already paid 74.5 percent of the finance charge and those are dollars you'll never see again.

How could this be? You thought that half the loan period had expired and, therefore, that half the interest was paid. Wrong—now you hear about the Rule of 78 which permits lenders to charge a much larger proportion of interest up front.

Background

Consumer loans, or installment loans as they are often called, usually have the interest as an add-on. This means that the full rate of interest is charged on the entire principal for the full loan period. For example, on a $4000 loan that is to run 36 months at nine percent interest, the interest would be $1080 ($360 per year for three years); therefore the amount to be repaid would be $5080, which would be divided into 36 equal monthly payments of $141.11. (Federal regulations require the lender to compute and disclose the Annual Percentage Rate (APR) to the borrower, which shows the actual rate being paid due to the decreasing balance of the loan. The APR will be significantly larger than the simple rate used to compute add-on interest as in the example above.)

The Rule of 78 is a frequently-used means of apportioning interest and principal in each monthly payment. As time passes during the life of the loan, the amount of interest paid each *calendar* year under this rule must be known by the borrower if he is to use it as an income tax deduction. If the loan runs full term, then nothing more needs to be known about the appor-

tionment. However, if the borrower decided to pay off the loan early, then the amount of interest paid up to that time is required so the rebate or unearned portion of the interest can be deducted by the lender from the loan total to determine the balance due at payoff time.

What does all this have to do with the number 78? The Rule of 78 is actually defined as a sum-of-digits method of calculation. The 78 comes from its application on loans lasting one year. If the sequential numbers from one to 12 are assigned to the 12 months in a year, the sum is 78: 12 + 11 + 10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 = 78. Obviously, the name of the rule is a misnomer because the method is used for a variety of loan periods longer and shorter than one year.

The theory behind this method is that the borrower on a one-year loan has 12 times as much of the debt outstanding at the time of the first installment as he would have at the last installment; 11 times as much at the second installment; 10 times at the third, and so on. Thus, he should be charged 12/78ths of the interest on the first installment; 11/78ths on the second; 10/78ths on the third, etc. Since the lender's administrative costs for a loan are largest when the loan is taken out, he can recover these costs, nearer to the time they were incurred using the Rule of 78.

Carrying this on when there is an early payoff, that portion of interest originally allocated to the time period after the payoff would not be charged. Thus, in the one-year loan case, if it were paid off in the 10th month, then the 11th and 12th month charges would not be made. They would be 2/78ths and 1/78th of the interest, or a total of 3/78ths would be rebated.

If a loan runs for longer than a year, the pattern is the same. For example, the sum of the months for two years is 300 and for three is 666.

Calculation

In real life things get more complicated than in the examples above. There might be a three-and-one-half year loan beginning in September with a finance charge of $1466.83 and paid off in November of the third year. The problem is to determine how much interest is paid in each *calendar* year and the amount of rebate at early payoff.

This calculation can be made by laboriously adding up the numbers representing each of 42 months and those representing the number of months that remain after the first year and after the second year. Then the number of months remaining after November of the third year must be determined; finally, all of these have to be manipulated into answers. Fortunately, mathematicians have come to the rescue and provided a formula for this kind of arithmetic progression. The formula tells us that $n(n + 1)/2$, where n is the number of months, will give the answer. Don't believe it? Try for one year: $12(12 + 1)/2 = 78$ which is the same total arrived at after add-

ing 12 + 11 + 10...

Given this helping hand, how do we find out how much the income tax deduction will be in the first year in the above example? First, determine what part of the total interest is allocable to payments made in the first year. This is: the sum of month numbers in entire loan period minus the sum of month numbers in period remaining after first year, all divided by the sum of month numbers in the entire loan period. Applying this: (903 – 741)/903 = .1794 or 17.94 percent of interest is paid in first calendar year.

Where did those numbers come from?

- Loan life is three and one-half years or 42 months:

$$n(n + 1)/2 = 42*43/2 = 903$$

- Payments began in September, so four payments were made in the first calendar year and 38 payments remain:

$$n(n + 1)/2 = 38*39/2 = 741$$

Applying the 17.94 percent to the total loan interest of $1466.83, we find interest for the first year is $263.15. This can then be entered on Schedule A of Form 1040 when the tax return is prepared.

Similarly, using the sum of month numbers remaining after the second year, we find that 43.19 percent of the total interest, or $633.52, would be paid and would be a tax deduction in the second year.

For the third year, when there is an early payoff in November, the formula is: the sum of month numbers remaining after the second year minus the sum of month numbers remaining after payoff, all divided by the sum of month numbers in the entire loan period. Applying this: (351 – 120)/903 = .2558 or 25.58 percent of total interest. Thus, $375.22 will be paid the third year.

Where did those numbers come from?

- At the end of the second year, the loan had 26 months to run (42 – 4 – 12 = 26):

$$n(n + 1)/2 = 26*27/2 = 351$$

- At payoff time the loan had 15 months to run (42 – 4 – 12 – 11 = 15):

$$n(n + 1)/2 = 15*16/2 = 120$$

Now, to determine the rebate:

- Rebate = Total interest less interest until payoff
- Rebate = 1466.83 – 263.15 – 633.52 – 375.22 = $194.94

If we had merely wanted to know the rebate to decide if it was worthwhile to pay the loan early, we could divide the sum of month numbers remaining after payoff by the sum of month numbers in the entire loan period. From numbers used above, this would be: 120/903 = .1329 or 13.29 percent of total interest or $194.94, the same as above.

Calculations like these are a real time-consuming pain and an invitation to errors. Sure, there are tables available to help with some of this, but

usually they are for specific loan lives that do not fit all cases. Also, early payoff tables are hard to find (see bibliography if you wish to pursue further). Why suffer through all this when your friendly computer is sitting in the corner begging to tackle a calculation like this—error free!

Take the time to copy the program listing. Put it in your library and you'll never have to calculate by the Rule of 78 again. The program is written in TRS-80 Level II BASIC. Program lines have been numbered in steps of 10 to facilitate use of the AUTO command in copying it. (Instant Software's Renumber program is a great tool for converting your programs to even steps this way.)

When entering program lines where portions are enclosed in quotes and on the DATA line 470, be certain to include spaces exactly as shown to assure proper format alignment. For example, in line 130 there are five spaces between the first quote mark and the word HOW. This spacing is particularly critical in lines 470, 490, and 500. Proper columnar alignment, as shown in figures 3 and 4, is determined by line 470. In this line there must be a total of eight letters and spaces for each word before the comma. For example FIRST_ _ _, SECOND_ _, THIRD_ _ _, etc.

Who thought up this rule? I have not been able to find out. References to the Rule of 78 were encountered in publications dated as far back as 1948, but there was no indication of its origin.

No doubt all this is much more than you ever thought you wanted to know about the Rule of 78, but at least if you did not know it before, you can avoid wrong assumptions about interest payments on installment loans. If you use the Rule of 78 in your profession, you can be assured of quick and error free calculations with this program.

Running the Program

Upon entry of brief information about the loan, the program provides the amount of interest and percent of total interest for each calendar year in the life of the loan or until the early payoff. For an early payoff it also shows the amount of rebate.

To input data for the program, the four questions shown in Figure 1 are answered. If the early payoff is answered YES, a fifth question about time of the payoff is asked, as shown in Figure 2.

HOW MANY MONTHLY PAYMENTS IN LIFE OF CONTRACT?_
IN WHAT MONTH DOES PAYMENT BEGIN?_
HOW MUCH IS TOTAL FINANCE CHARGE (INTEREST) IN DOLLARS AND CENTS?_
WAS THERE AN EARLY PAYOFF OF THE LOAN CONTRACT?_

Figure 1

The program then lists the calculated answers and summarizes original input data above the listing. An example run for a five-year loan that goes to maturity is shown in Figure 3. Figure 4 shows the same loan with an early payoff after 38 months.

```
HOW MANY PAYMENTS ARE THERE UP TO AND INCLUDING MONTH OF THE
EARLY PAYOFF?_
```

Figure 2

```
PMTS: 60 MOS.   1ST PAY: AUGUST.   EARLY PAYOFF: NONE.
YEAR              PERCENT              INTEREST
FIRST             15.85                $375.48
SECOND            32.46                $769.08
THIRD             24.59                $582.63
FOURTH            16.72                $396.19
FIFTH              8.85                $209.75
SIXTH              1.53                 $36.25
                  TOTAL INTEREST IS: $2,369.38
```

Figure 3

As written, the program limits loan life to 120 months (10 years). Most loans of this type are shorter. Correcting messages are provided if absurd entries are made such as loan life of one month or less and early payoff time in excess of loan life. If negative amounts are entered, they are made positive. Life of loan entries are converted to integers if they are entered with decimal amounts (e.g., 24.6 months will be calculated as 24 months). If the number of months for early payoff is entered with decimal amounts, a choice is given

```
PMTS: 60 MOS.   1ST PAY: AUGUST.   EARLY PAYOFF: NONE.
YEAR              PERCENT              INTEREST
FIRST             15.85                $375.48
SECOND            32.46                $769.08
THIRD             24.59                $582.63
FOURTH            13.28                $314.62
                  TOTAL INTEREST IS: $2,041.81
THIS IS 86.17% OF TOTAL FINANCE CHARGE OF $2,369.38
DUE TO EARLY PAYOFF. THUS 'REBATE' IS: $327.57
```

Figure 4

for rounding up to a full month or re-entering. The program accepts month of first payment input as the full name of the month or as an abbreviation of the month (e.g., AUG; at least three letters). The program accepts finance charges in any amount, but there is some rounding of amounts over $10,000, and some leading percent signs will show if finance charges exceed $100,000 due to use of the PRINT USING statement. However, dollar amounts will be correct. Calculations are single-precision which is more than adequate for most loans of this type.

Extending the Program

Most installment loans using the Rule of 78 are for periods shorter than the 10 year maximum provided in the program. If loan lives longer than 10 years are desired, the program can be modified by changing lines 80, 150, 470, 480, and 750. Depending on how much longer than ten years, additional lines equivalent to 490 and 500 may need to be added to control scrolling.

Spacing in all data entries on Line 470 will have to be adjusted if the program is to include a thirteenth year or more. Otherwise, format of the run will not be uniform. Note that because the program provides for the loan to begin at any month during the year, provision in lines 470 and 480 have to be made for one more calendar year than the maximum number of years allowed for life of the loan.

For reference, Table 1 shows a list of variables used in the program.

Most Significant

N	Number of payments in loan life
M$	Month of first payment
P	Payments in first year
T & TI	Total interest
E$	Early payoff question response
D	Divisor in sum of months formulas
PO	Payoff payments
PR	Payments remaining after payoff
I	Interest
PO$	Payoff question response about decimal months

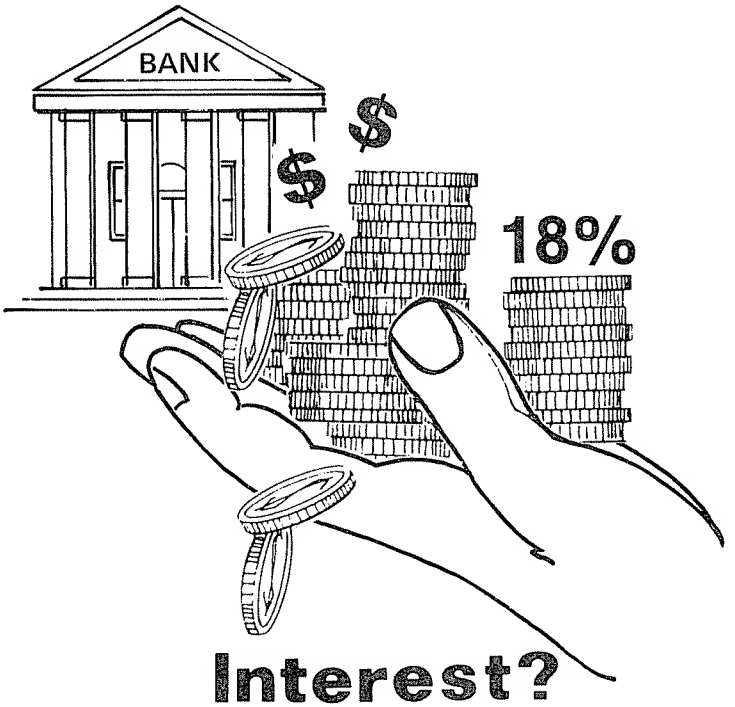
Others used

O,OO,Z,G,B,E,X,A$,TI$,YP$,Z$,UP$,UI$

Table 1. *List of Variables*

Bibliography

- Greynolds, E. B., Jr., Aronofsky, J. S., Frame, R. J., *Financial Analysis Using Calculators*, McGraw-Hill, N.Y., NY, 1980.
- More, N. D., *Dictionary of Business Finance & Investment*, Investors Systems, Inc., Dayton, OH, 1975.
- Neifeld, M. R., *Neifeld's Guide to Installment Computations*, Mack Publishing Co., Easton, PA, 1953. (Has early payoff table for selected periods up to 36 months)
- Swindle, R. E., *Business Math Basics*, Wadsworth Publishing Co., Belmont, CA, 1979.
- Webber, R. P., *Business Math—A Consumer Approach*, Houghton Mifflin, 1976.
- *New Rules on Consumer Credit Protection*, Commerce Clearing House, N.Y., NY, 1969.



Program Listing

```
10 CLS
20 PRINT CHR$(23)
30 PRINT @ 322,"* * * * RULE OF '78 * * * *"
40 FOR O = 1 TO 1200:
    NEXT O
50 CLS :
   PRINT :
   PRINT "THIS IS A PROGRAM TO CALCULATE PERCENT AND AMOUNT OF INTE
   REST"
60 PRINT "PAID EACH YEAR WHEN A LOAN IS BEING PAID OFF UNDER THE 'R
   ULE "
70 PRINT "OF 78'.  MONTHLY INSTALLMENTS MUST BE EQUAL DOLLAR AMOUNT
   S AND
80 PRINT "LIFE OF THE LOAN CONTRACT MUST NOT EXCEED 120 MONTHS (10
   YEARS).IF THERE IS AN EARLY PAY-OFF OF THE LOAN, PROGRAM WILL AL
   SO     ADJUST INTEREST ACCORDINGLY.
90 FOR OO = 0 TO 2000:
    NEXT OO
100 PRINT :
   PRINT "JUST ENTER THE FOLLOWING INFORMATION ABOUT THE LOAN:"
110 PRINT :
   CLEAR
120 DEFINT N
130 INPUT "      HOW MANY MONTHLY PAYMENTS IN LIFE OF CONTRACT ";N
140 IF N < 2
    THEN
     730
150 IF N > 120
    THEN
     750
160 PRINT :
   INPUT "      IN WHAT MONTH DOES PAYMENT BEGIN ";M$
170 IF LEFT$(M$,3) = "JAN" P = 12
180 IF LEFT$(M$,3) = "FEB" P = 11
190 IF LEFT$(M$,3) = "MAR" P = 10
200 IF LEFT$(M$,3) = "APR" P = 9
210 IF LEFT$(M$,3) = "MAY" P = 8
220 IF LEFT$(M$,3) = "JUN" P = 7
230 IF LEFT$(M$,3) = "JUL" P = 6
240 IF LEFT$(M$,3) = "AUG" P = 5
250 IF LEFT$(M$,3) = "SEP" P = 4
260 IF LEFT$(M$,3) = "OCT" P = 3
270 IF LEFT$(M$,3) = "NOV" P = 2
280 IF LEFT$(M$,3) = "DEC" P = 1
290 IF P = 0 PRINT :
   PRINT "PLEASE ENTER MONTH OR CORRECT 3 LETTER ABBREVIATION FOR M
   ONTH.":
   GOTO 160
300 IF P > N
    THEN
     P = N
310 PRINT :
   INPUT "      HOW MUCH IS TOTAL FINANCE CHARGE (INTEREST) IN DOLLA
   RS           AND CENTS";T
320 T = ABS(T)
330 PRINT :
   IF T > 9999.99 PRINT "(FOR FINANCE CHARGES EXCEEDING $10,000, MI
   NOR ROUNDING ERRORS    MAY OCCUR IN INTEREST DOLLAR AMOUNTS.)":
   PRINT :
    ELSE
     350
340 FOR Z = 1 TO 1500:
    NEXT Z
350 INPUT "      WAS THERE AN EARLY PAY-OFF OF THE LOAN CONTRACT";E$
360 IF E$ = "YES" GOTO 770
370 IF E$ < > "NO" PRINT "PLEASE ANSWER YES OR NO.":
   GOTO 350
```

```
380 CLS
390 D = N * (N + 1) / 2
400 G = N - P
410 IF PR > 0 AND G < PR
     THEN
      G = PR
420 E = D
430 GOSUB 680
440 PRINT "PMTS:"N"MOS.";"    1ST PAY: "M$".";:
    IF PO = 0 PRINT "     EARLY PAY-OFF: NONE." :
     ELSE
      PRINT "     EARLY PAY-OFF:"PO"MOS."
450 PRINT TAB(8)"YEAR"; TAB(25)"PERCENT"; TAB(44)"INTEREST"
460 PRINT TAB(7) STRING$(7,129); TAB(24) STRING$(10,129); TAB(43)
    STRING$(11,129)
470 DATA "FIRST   ","SECOND  ","THIRD   ","FOURTH  ","FIFTH   ","SIX
    TH    ","SEVENTH ","EIGHTH  ","NINTH   ","TENTH   ","ELEVENTH
480 FOR X = 1 TO 11:
     READ A$
490  IF PR > 0 AND A$ = "EIGHTH  " GOSUB 870
500  IF PR = 0 AND A$ = "ELEVENTH" GOSUB 870
510  PRINT TAB(7) A$;:
     GOSUB 830
520  GOSUB 640
530  NEXT X
540 TI$ = "$$#,###.##"
550 PRINT TAB(24)"TOTAL INTREST IS:"; PRINT USING TI$;TI
560 IF T = 0 GOTO 590
570 YP$ = "###.##"
580 IF E$ = "YES" PRINT :
    PRINT "     THIS IS";:
    PRINT USING YP$;100 * TI / T;:
    PRINT "% OF TOTAL FINANCE CHARGE OF";:
    PRINT USING TI$;T;:
    PRINT " DUE" :
    PRINT "     TO EARLY PAY-OFF.  THUS 'REBATE' IS:";:
    PRINT USING TI$;T - TI
590 PRINT :
    INPUT "DO YOU WANT TO RUN ANOTHER CALCULATION";Z$
600 IF Z$ = "YES"
     THEN
      CLS :
      GOTO 110
610 IF Z$ = "NO"
     THEN
      CLS :
      PRINT :
      PRINT @405, "THANK YOU, GOODBYE.":
      PRINT :
      PRINT :
      GOTO 630
620 PRINT :
    PRINT "PLEASE ANSWER YES OR NO.":
    GOTO 590
630 END
640 G = G - 12
650 IF PR > 0 AND G < PR
     THEN
      G = PR
660 IF G < 0
     THEN
      G = 0
670 E = B
680 B = G * (G + 1) / 2
690 I = (E - B) / D
700 TI = TI + I * T
710 IF I = 0
     THEN
      540
```

Program continued

```
720 RETURN
730 PRINT :
    INPUT "LIFE OF CONTRACT CAN'T BE UNDER 2; CORRECT LIFE";N
740 GOTO 140
750 CLS :
    PRINT :
    PRINT :
    PRINT "SORRY, LIFE OF CONTRACT IN THIS PROGRAM IS LIMITED TO 120
     MONTHS(10 YEARS).  TRY AGAIN."
760 PRINT :
    GOTO 110
770 PRINT :
    INPUT "     HOW MANY PAYMENTS ARE THERE UP TO AND INCLUDING MONT
    H OF        THE EARLY PAY-OFF";PO
780 PO = ABS(PO)
790 IF PO = > N CLS :
    PRINT :
    PRINT :
    PRINT :
    PRINT :
    PRINT :
    PRINT "     PAYMENTS MADE UNTIL EARLY PAY-OFF CAN'T EQUAL OR EXC
    EED         LIFE OF CONTRACT; TRY AGAIN.":
    GOTO 770
800 IF PO > INT(PO) GOSUB 890
810 PR = N - PO
820 GOTO 380
830 UP$ = "          ###.##"
840 UI$ = "           $$#,###.##"
850 PRINT USING UP$;I * 100;:
    PRINT USING UI$;I * T
860 RETURN
870 PRINT "NOTE-TOP INFO IS GOING TO DISAPPEAR; PRESS ANY KEY TO CON
    TINUE."
880 IF INKEY$ = "" GOTO 880 :
     ELSE
      RETURN
890 CLS :
    PRINT :
    PRINT :
    PRINT "UNDER RULE OF 78 IF THE NUMBER OF MONTHS UNTIL PAY-OFF IN
    CLUDES A PORTION OF A MONTH, IT IS COUNTED AS A FULL MONTH.  YOU
    R ENTRYOF"PO"MONTHS WILL BE CHANGED TO" INT(PO) + 1"FOR THE CALC
    ULATION."
900 PRINT :
    PRINT "IF THIS IS OK, ENTER YES; IF NOT, ENTER NO TO RE-DO";:
    INPUT PO$
910 IF PO$ = "YES"
     THEN
      PO = INT(PO) + 1:
      GOTO 810
920 IF PO$ = "NO" GOTO 770
930 PRINT :
    PRINT "PLEASE ANSWER YES OR NO.":
    GOTO 900
940 RETURN
```

EDUCATION

Computer Education

Measuring Instructional Effectiveness with the TRS-80

Using a TRS-80 to Tabulate Student Ratings

EDUCATION

Computer Education

by Joseph R. Chartier W1RFE
and Carl A. Goldner W4ERA

Ken Vianello is the principal of the Fort King Middle School in Ocala, Florida, and not only does he believe in the importance of the computer and its impact on the future, but he also acts on his beliefs.

If anyone had told you ten years ago that an eighth grade class in computer education would be the most popular course at that grade level, you might have concluded he was a prime candidate for the funny farm. Not so! Here at Fort King, a computer education course in BASIC is, in fact, most popular with eighth graders.

The course is taught in a computer laboratory equipped with sixteen Level II TRS-80s. Perhaps even more amazing than the fact that such a facility *does* exist in a middle school is the story of how the lab came to be.

Spice Sale

Shortly after his appointment as principal of this middle school (grades 6 through 8), Ken Vianello, convinced that computers were here to stay, laid the groundwork to prepare students for the impact of these devices.

As Vianello observed, "We in education are notorious for reacting but never acting. If the prediction is true that over 40 million microcomputers will find their way into our homes in the next decade, then it makes good sense to start now to train our young people in the practical applications of this equipment."

The funds for this project were not available through regular budget channels, so Vianello elected to raise the money through the sale of spices. With the enthusiastic support of the parents and students, some $19,000 worth of "sneeze-proof" pepper, "cry-proof" onions, seasoning salt, and bacon bits were sold, providing sufficient profits to finance the program.

In the early part of 1979, with the spice drive funds in hand, Principal Vianello, hopeful that the program could be operational for the 1979-80 school year, assembled his staff, Ms. Jane McClellan, curriculum supervisor, and Ms. Holley Griffin, a math teacher with a flair for computing.

They purchased fifteen Level II 4K TRS-80 units after reviewing most of the microcomputer equipment that could do the job within their budget limitations. Renting, leasing, and time-sharing arrangements were considered and subsequently discarded, since future budget appropriations could jeopardize the program.

Computer Carpentry

The Fort King Middle School has an enrollment of 1100 with a faculty and staff of 57, yet they found time to build a computer lab. Yes, it was built from scratch—for where else could one find an eighth grade computer laboratory? Moreover, the only space available to house the facility was a multi-level lecture hall that required considerable carpentry work to convert it into a flat-floored lab. Virtually all the lecture hall furniture was adapted for use as computer work desks, saving considerable expense.

If any of the staff were skeptical, they have long since joined the ranks of the believers.

Let's make one point very clear: The course is not elective—it is required of every eighth grade student! And there are 350.

Each student receives eighteen weeks of instruction, in two nine-week segments, separated by a nine-week break. With five 50-minute periods per week, the course provides nearly 75 hours of classroom time.

Despite the fact that the course is required, to date, Vianello reports, only five youngsters have been lukewarm to cool in their reactions to the training, and only two of these asked to be excused.

For the majority of students, the course has been a success. School starts at 8:00 am. Ms. Holley Griffin, who teaches five of the daily classes, usually arrives about 7:00 am, hoping for a few minutes alone at her newly acquired TRS-80 Level II with disk drive. Invariably, two or three youngsters are already there waiting for her to open the lab. By 7:30 most of the first class of the day is on hand with all keyboards in use. After the final class of the day, there are some enthusiasts who must be literally pried loose from the keyboard.

Why is the course so successful?

It's probably a variety of things, not the least of which are good teaching methods coupled with high student interest. Classroom work is assigned according to the student's capability. In general, two students of comparable ability share the use of a TRS-80.

Grading is on the satisfactory/not satisfactory basis, so none of the students are made to feel they are struggling for a particular grade.

Learning Levels

While most of the pupils are permitted to set their own pace, the high achievers are given additional classroom work and homework on a much tighter and structured schedule. This group is required to turn in fully documented special assignments at least every two weeks.

Ms. Griffin and Ms. McClellan combined their talents to prepare some project material tailored to the three student learning levels: the gifted, the average, and the low achievers. Space does not permit a complete list, but a

few from each category are outlined below.

1. Gifted or Advanced Student Projects.

- Write a program which asks a person his or her weight; print that person's equivalent weight on each of the planets and the moon.
- Develop a program which translates an alphabetic message into Morse code.
- If you put P dollars into a savings account with an interest rate R, compounded T times a year, how much money would there be in the account at the end of N years? Write the program so that it answers the question for any combination of the parameters P, R, T, and N. Now use the program to determine the amount A after one year, starting with $100 at five percent interest when it is compounded (1) annually, (2) semi-annually, (3) quarterly, and (4) daily.
- You are about to purchase a car. Assume you normally drive 10,000 miles per year and use EPA mileage ratings from 10 miles per gallon (MPG) for a heavy luxury car to 40 MPG for a small economy sedan. Write a program which lists, in three columns, the MPG from 10 to 40, the gallons of gasoline used in one year, and how much that gas costs using a current local price for unleaded gas.

2. Average Student Projects.

In this category Ms. Griffin has provided eight pages of projects and problems to give the students some good programming practice. Students are asked questions such as:

- How would you correct it to make it run right?
- How would you modify it to make it a better program?
- Compare two programs shown; tell which you prefer and why.
- Take a listed program and make your own adaptation.

The program guide provides a number of examples to apply to the above practice work.

A section in the Average Student Guide asks the student to translate some word problems into BASIC programming formats. They are asked to copy the finished program on a BASIC Coding Sheet, remembering the steps for developing a program: (1) Feed in the data, (2) Provide an equation (formula), and (3) Print the answer.

Most of this exercise, for the average student, involves taking word problems in arithmetic and converting them into computer language. For example:

- A carton of soft drinks costs 98¢, and a doughnut costs 12¢. What is the total cost of three cartons of soft drinks and ten doughnuts?
- A family drove 2,300 kilometers one summer. The next summer, they drove 1,084 kilometers. How much further did they drive the first summer?

3. Projects for Low Achievers.

For this group seven pages of simple program exercises for copy practice on the TRS-80 keyboard have been prepared. These are some examples:

```
10 LET X = 100/4 + 75
20 LET Y = 200/20
30 PRINT X/Y
40 END
```

```
10 REM * ADDITION PROBLEM *
20 READ A,B,C,D
30 LET E = A + B + C + D
40 PRINT E
50 DATA 25, 3, 17, 12
```

Teacher's note: Remember how we got rid of the "?O.D. error" in another program we did?

```
10 PRINT " MY COMPUTER IS A WHIZ AT ARITHMETIC"
20 PRINT 5 + 2 * 4 + 3
30 PRINT 8 - 16/32
40 PRINT (5 + 2) * (8 - 3)
50 PRINT "THAT'S ALL FOLKS!"
60 END
```

It was this third group that surprised school authorities. These youngsters have an extremely short attention span, and they cause discipline problems in many classrooms. Not so here in the computer lab!

When one of these students sat down at the TRS-80 keyboard, he seemed transformed into a totally different student. His interest was quite intense in the short programs he was given to copy.

After copying a number of simple programs, some of these pupils even dream up programs of their own. The following program was written by one of the youngsters:

```
10 FOR I = 0 TO 1332
20 X = RND (1023)
30 PRINT X,"*";
40 NEXT I
```

Teaching Technique

Ms. Griffin's expertise in math, coupled with some computer training, made her an ideal choice for her position. In addition to some of the teaching material she has developed herself, she makes liberal use of various TRS-80 manuals including *Learning Level II* by David A. Lien. The lab's reference shelves contain a number of publications which the students can consult at any time. In the laboratory, Ms. Griffin has the usual chalkboard and overhead projector which are used to explain and demonstrate various steps in BASIC programming. Considerable material is on transparencies and can

be readily copied by students on their keyboards from the projection screen.

The Computer Education Program has been well received and enjoyed by the students, and there have been few disciplinary problems. Interest in the course is spreading in a contagious way! Members of the faculty, parents, and other outsiders have indicated a desire to learn more about microcomputers as a direct result of student enthusiasm.

At least one adult education class is using the laboratory for an evening computer course in BASIC. There have even been inquiries from county and city service departments concerning training programs for employees. If the demand persists, Ken Vianello's ingenuity may well be tried again.

What about Service?

What about equipment problems?

In general, the staff feels they have received good service and support from the two Radio Shack stores in Ocala.

Local store managers have graciously cooperated by loaning a keyboard or two during repairs. No serious interruptions have occurred because of equipment failures.

What would they change, if they had it to do again? "Very little," said the Curriculum Supervisor, Jane McClellan.

Some of the students doubted they would have chosen computer ed if it had been an elective subject, but now that they are immersed in it, you couldn't pry them loose. Sixth and seventh graders have shown considerable interest in the course and are impatient to get "with it" in the next year or two.

Several Ocala families have purchased microcomputers for the home as a direct result of the interest sparked by this forward-looking program at the Fort King Middle School.

An adult class is using the lab for a Central Florida Community College course in BASIC, and many of them have microcomputers on order. The senior members of the class are just as enthusiastic as the younger members.

The use of the microcomputer as a patient teaching aid has just begun. Youngsters are able to accept that the computer cannot forgive mistakes and will not tolerate sloppy or faulty instructions. Students have, in exasperation, called the machine an idiot or a dumbbell, but they do not get mad at the TRS-80 when it tells them they are wrong.

Progressive educators like Ken Vianello and Jane McClellan are providing the direction and leadership for this innovative experience in education.

Teachers Tommy Parker, who works with a group of gifted youngsters, and Holley Griffin are, in a real sense, pioneers in this field, and their contribution will not go unrecognized. As a result of the excellent work of this group and the efforts of others in this dynamic venture, we can look forward to an interesting future with the computer as a willing help-mate.

Measuring Instructional Effectiveness with the TRS-80

by Maj. Vernon Humphrey

Most microcomputer applications in the field of education involve either using the computer as an instructional device or using it for test grading and similar administrative purposes. One of the most fruitful applications of the computer is often overlooked: using the computer as a tool to measure the effectiveness of instruction and to provide a basis for improving the instruction.

There are many methods of designing good tests, but no test, no matter what theory the instructor follows, can be regarded as effective until it has been validated. One of the most valuable tools in validating tests is the PHI coefficient. It provides a statistical basis for determining the quality and effectiveness of the test.

There are two basic methods of validating tests, both of which use the PHI coefficient. Depending on the type of course you are teaching, you may choose to use one or both methods.

One of the most critical factors in teaching is constructing good tests. From the students' standpoint, a well-designed test is an assurance that the conscientious student will be properly rewarded for his efforts with a good test score. For the teacher, the test forms the basis for improvements in course content and teaching techniques. In industrial training programs, employers and supervisors need assurance that persons who have been trained can do the jobs they were trained for.

All too often, however, an instructor, after putting in many long hours in preparing lesson plans and teaching classes, puts together a test which really doesn't measure anything.

The first method measures the validity of the test as a predictor of on-the-job effectiveness. The instructor designs a test which he hopes will measure job mastery and administers the test to two groups. The first group consists of people who are known to be proficient in the job (this can be determined by querying supervisors, consulting productivity records, or direct observation at the job site). The first group is called the "master" or "instructed" group. The second group ("non-master" or "non-instructed") consists of people who are known *not* to be proficient in the job. The test is administered to both groups and the PHI coefficient is used to determine which questions or test elements adequately discriminate between groups. The test is modified until all questions or test elements provide a good level of discrimination.

The second method is to validate the test for use as a yardstick to measure the effectiveness of the instruction, in order to improve the instruction. Again, two groups are used. The "instructed" group is given the instruction before the test is administered. The "non-instructed" group is given the test without the instruction.

In both methods, each question is scored as "pass" or "fail." A correct response constitutes a "pass" and an incorrect response, regardless of the number of possible responses, is scored as "fail." The PHI coefficient calculation compares the responses of the two groups on a question-by-question basis. It pinpoints those questions which do not discriminate between groups well enough to be useful in predicting job performance or in measuring teaching effectiveness.

The PHI coefficient also indicates the reasons for inadequacy and suggests corrective action. Since there are two groups tested and two possible outcomes (pass or fail) for each question, there are four possible situations for each question:

1. Instructed students tend to pass and non-instructed students tend to fail. This indicates a valid question and the PHI coefficient will be +0.3 or higher.

2. Both groups tend to pass. This indicates the question is either "transparent" or covers something that is common knowledge. It does not adequately discriminate and should be revised, eliminated, or the scoring weight should be reduced. The PHI coefficient will be less than +0.3.

3. Both groups tend to fail the question. If you are validating to predict job performance, this question does not discriminate and should be eliminated. If you are validating the test for use as a yardstick, the instruction itself is at fault. The PHI coefficient will be less than +0.3.

4. Instructed students tend to fail and non-instructed students tend to pass. Although this kind of question does discriminate, it indicates fundamental shortcomings in design. Either the actual job practice is different from what the instructor or supervisor believes it to be, or the instruction is confusing or wrong. The PHI coefficient will be strongly negative.

Running the Program

The program will run in 16K Level II. It allows you to define the size of the test by entering the number of questions in the test and then accepts question-by-question input for instructed and non-instructed students. The two groups must be equal in size and the program tests for group size quality. Data can be entered by keyboard or tape.

The program asks you to enter PASS (1) or FAIL (2) for each question for each student. You terminate entries by entering 99999. The program will accept only those three numbers, so if you have a problem with keybounce, you won't spoil your data.

When you have terminated entries for the instructed group, the program calls for the non-instructed group. The entries will automatically terminate when the numbers of the two groups are equal. You can also terminate non-instructed entries with 99999. In that case the two groups will not be equal, of course, and the program will give you the choice of adding more non-instructed students, deleting the extra instructed students, or aborting the run.

When you have equal numbers in the two groups, press ENTER to continue the program. To read or edit the data, select the READ option. If you want to record the data, you must read before recording.

Both READ and CALCULATE PHI COEFFICIENT options give you a choice of video or print output. Video output displays a screenful at a time in READ and a question at a time in CALCULATE options. The RECORD option uses the high-speed tape recording routine published in the July 1979 issue of the Radio Shack *Microcomputer Newsletter*. Lines 1550 and 1560 are a short tape manipulating routine originally published in the November 1978 issue.

As written, the program will accept up to 600 responses for each group of students. If you have a test with twenty questions, you can have up to thirty students in each group. For larger numbers, redimension arrays A and B in line 10.

Program Listing

```
10 CLEAR 1200:
   DIM A(600):
   DIM B(600)
20 A = 0:
   B = 0:
   C = 0:
   D = 0:
   J = 1:
   K = 1
30 :
   '  PHI Coefficient Program    by   MAJ V. Humphrey,   US Army In
   stitute for Professional Development,    Ft. Eustis, VA   23604
40 CLS :
   PRINT :
   PRINT :
   PRINT STRING$(64,"%"):
   PRINT :
   PRINT TAB(18)"PHI   C O E F F I C I E N T"
50 PRINT :
   PRINT :
   INPUT "            1=KEYBOARD INPUT, 2=TAPE INPUT. WHICH";G:
   ON G GOTO 60,930
60 PRINT :
   PRINT :
   INPUT "HOW MANY QUESTIONS IN EXAM";N
70 PRINT "BEGIN  ENTERING  DATA  FOR INSTRUCTED STUDENTS. 1=PASS, 2
   =FAIL. (ENTER 99999 TO END.)"
80 PRINT "STUDENT NUMBER"(J - 1) / N + 1
90 FOR I = 0 TO N - 1
100  PRINT "QUESTION"I + 1
110  INPUT A:
     IF A = 99999
      THEN
       150
120  IF A < > 1 AND A < > 2
      THEN
       INPUT "ERROR! 1=PASS, 2=FAIL. PRESS 'ENTER' WHEN READY TO CON
       TINUE";E:
       GOTO 110
130  A(J + I) = A
140  NEXT I:
    J = J + N:
    PRINT :
    GOTO 80
150 CLS :
    PRINT "BEGIN ENTERING DATA FOR NON-INSTRUCTED STUDENTS. (ENTER 9
    9999 TO END)."
160 PRINT "STUDENT NUMBER"(K - 1) / N + 1
170 FOR I = 0 TO N - 1
180  PRINT "QUESTION";I + 1
190  INPUT A:
     IF A = 99999
      THEN
       250
200  IF A < > 1 AND A < > 2
      THEN
       INPUT "ERROR! 1=PASS, 2=FAIL. PRESS 'ENTER' WHEN READY TO CON
       TINUE";E:
       GOTO 190
210  B(K + I) = A
220  NEXT I:
    K = K + N:
    PRINT :
    IF K = J
     THEN
      240
```

Program continued

```
230 GOTO 160
240 INPUT "NUMBER  OF  NON-INSTRUCTED STUDENTS EQUALS NUMBER OF INST
    RUCTED STUDENTS. PRESS 'ENTER' WHEN READY TO CONTINUE.";E:
    ON E GOTO 300
250 IF K = J
     THEN
      300
260 PRINT "YOU HAVE "(J - 1) / N" INSTRUCTED  AND "(K - 1) / N" NON-
    INSTRUCTED  STUDENTS.  yOU MUST HAVE EQUAL NUMBERS."
270 INPUT "1=DELETE EXTRA STUDENTS. 2=ADD NEW STUDENTS. 3= ABORT.WHI
    CH";E:
    ON E GOTO 280, 1540, 1570
280 IF K > J
     THEN
      K = J
290 J = K
300 CLS :
    INPUT "1=READ. 2=CALCULATE PHI COEFFICIENT. WHICH";E:
    ON E GOTO 310,1150
310 INPUT "PRINT OR VIDIO (P/V)";Z$:
    IF Z$ = "P" GOTO 1380
320 CLS :
    E = 1:
    PRINT "FOR INSTRUCTED STUDENTS"
330 FOR I = E TO J - 1 STEP N
340  PRINT "STUDENT NUMBER " INT(I / N) + 1
350  FOR H = 0 TO N - 1
360   PRINT TAB(5)A(I + H);
370   NEXT H:
     PRINT
380  IF I = E + 4 * N
      THEN
       E = E + 4 * N:
       GOTO 400
390  GOTO 410
400  INPUT "WANT TO CONTINUE";G
410  NEXT I
420 INPUT "NON-INSTRUCTED STUDENTS";E:
    ON E GOTO 430
430 CLS :
    E = 1
440 FOR I = E TO K - 1 STEP N
450  PRINT "STUDENT" INT(I / N) + 1
460  FOR H = 0 TO N - 1
470   PRINT TAB(5)B(I + H);
480   NEXT H:
     PRINT
490  IF I = E + 4 * N
      THEN
       E = E + 4 * N:
       GOTO 510
500  GOTO 520
510  INPUT "WANT TO CONTINUE";G
520  NEXT I
530 INPUT "1=READ. 2=CHANGE LINE. 3=DELETE STUDENT. 4=ADD STUDENT.5=
    RECORD. 6=CALCULATE  PHI.WHICH";G
540 ON G GOTO 310, 550, 670, 1540, 780, 1150
550 INPUT "1=INSTRUCTED STUDENTS. 2=NON-INSTRUCTED STUDENTS. WHICH";
    L
560 INPUT "WHICH STUDENT. (ENTER 99999 TO END)";M:
    IF M = 99999
     THEN
      530
570 FOR I = 0 TO J - I STEP N
580  IF I / N + 1 = M
      THEN
       600
590  NEXT I
600 FOR H = 0 TO N - 1
```

```
610  PRINT "QUESTION"H + 1
620  INPUT A
630  IF L = 1
      THEN
       A(I + H + 1) = A
640  IF L = 2
      THEN
       B(I + H + 1) = A
650  NEXT H
660 GOTO 550
670 INPUT "1=INSTRUCTED STUDENTS. 2=NON-INSTRUCTED STUDENTS. 3=RETUR
    N TO MAIN PROGRAM.WHICH";L:
    IF L = 3
     THEN
      530
680 INPUT "WHICH STUDENT. (ENTER 99999 TO END)";M:
    IF M = 99999
     THEN
      670
690 Z = (N * M) - (N - 1):
    J1 = J - N:
    J2 = J:
    IF L = 2
     THEN
      J1 = K - N:
      J2 = K
700 PRINT "NUMBER OF STUDENTS NOW = "(J1 - 1) / N
710 FOR I = Z TO J2 - 1 STEP N
720  FOR H = 0 TO N - 1
730   IF L = 1
       THEN
        A(I + H) = A((I + H) + N)
740   IF L = 2
       THEN
        B(I + H) = B((I + H) + N)
750   NEXT H:
     NEXT I:
    IF L = 1
     THEN
      J = J1
760 IF L = 2
     THEN
      K = J1
770 GOTO 670
780 INPUT "NAME FOR DATA FILE";D$
790 INPUT "TAPE SET TO RECORD";G
800 B$ = "":
    A$ = "/"
810 PRINT # - 1,D$,N,J,K
820 FOR I = 1 TO J
830  IF LEN(B$ + STR$(A(I)) + A$) > 230
      THEN
       PRINT # - 1,B$:
       B$ = ""
840  B$ = B$ + STR$(A(I)) + A$
850  NEXT :
    PRINT # - 1,B$
860 B$ = "":
    A$ = "/"
870 FOR I = 1 TO J
880  IF LEN(B$ + STR$(B(I)) + A$) > 239
      THEN
       PRINT # - 1,B$:
       B$ = ""
890  B$ = B$ + STR$(B(I)) + A$
900  NEXT :
    PRINT # - 1,B$:
    GOTO 530
910 NEXT I
```

Program continued

```
 920 GOTO 530
 930 INPUT "TAPE LOADED AND SET TO RUN";G
 940 A$ = "/":
     B$ = "":
     J = 1
 950 INPUT # - 1,D$,N1,NN,K
 960 PRINT "FILE BEING READ = "D$
 970 INPUT # - 1,B$:
     N = LEN(B$):
     LF = 1
 980 FOR I = 1 TO N
 990  IF MID$(B$,I,1) = A$
       THEN
        NC = I - LF :
       ELSE
        GOTO 1020
1000  A(J) = VAL( MID$(B$,LF,NC))
1010  LF = I + 1:
      J = J + 1
1020  IF J > NN
       THEN
        1040
1030  NEXT I:
     GOTO 970
1040 N = N1:
     J = NN
1050 A$ = "/":
     B$ = "":
     J = 1
1060 INPUT # - 1,B$:
     N = LEN(B$):
     LF = 1
1070 FOR I = 1 TO N
1080  IF MID$(B$,I,1) = A$
       THEN
        NC = I - LF :
       ELSE
        GOTO 1110
1090  B(J) = VAL( MID$(B$,LF,NC))
1100  LF = I + 1:
      J = J + 1
1110  IF J > NN
       THEN
        1130
1120  NEXT I:
     GOTO 1060
1130 N = N1:
     J = NN
1140 GOTO 530
1150 IF J < > K
      THEN
       260
1160 INPUT "PRINT OR VIDIO (P/V)";Z$
1170 CLS :
     H = 0:
     A = 0:
     B = 0:
     C = 0:
     D = 0
1180 FOR I = 1 TO J - 1 STEP N
1190  IF A(I + H) = 2
       THEN
        B = B + 1
1200  IF A(I + H) = 1
       THEN
        A = A + 1
1210  IF B(I + H) = 2
       THEN
        D = D + 1
```

```
1220  IF B(I + H) = 1
       THEN
        C = C + 1
1230  NEXT I:
     IF Z$ = "P"
      THEN
       1470
1240 PRINT :
     PRINT :
     PRINT "FOR QUESTION "H + 1:
     PRINT TAB(25)"PASS"; TAB(35)"FAIL":
     PRINT "INSTRUCTED"; TAB(25)A; TAB(35)B:
     PRINT "NON-INSTRUCTED"; TAB(25)C; TAB(35)D
1250 A2 = (A * D) - (B * C)
1260 B2 = SQR((A + B) * (C + D) * (A + C) * (B + D)):
     IF B2 = 0
      THEN
       1490
1270 X = A2 / B2
1280 IF Z$ = "P"
      THEN
       1480
1290 PRINT "PHI COEFFICIENT FOR QUESTION "H + 1" = "X
1300 H = H + 1:
     IF H > N - 1
      THEN
       1350
1310 A = 0:
     B = 0:
     C = 0:
     D = 0
1320 IF Z$ = "P"
      THEN
       1340
1330 INPUT "WANT TO CONTINUE";G
1340 GOTO 1180
1350 PRINT :
     PRINT :
     INPUT "LAST QUESTION. 1= RECALCULATE, 2=RETURN TO MAIN PROGRAM,
     3= END. WHICH.";G
1360 ON G GOTO 1150,530,1370
1370 END
1380 LPRINT "STUDENT RESPONSES FOR "D$:
     LPRINT :
     LPRINT :
     LPRINT "FOR INSTRUCTED STUDENTS":
     LPRINT
1390 FOR I = 1 TO J - 1 STEP N:
      LPRINT "STUDENT NUMBER " INT(I / N + 1)
1400  FOR H = 0 TO N - 1:
       LPRINT TAB(5)A(I + H);
1410   NEXT H:
      LPRINT :
      NEXT I
1420 LPRINT :
     LPRINT :
     LPRINT "NON-INSTRUCTED STUDENTS"
1430 LPRINT :
     FOR I = 1 TO K - 1 STEP N:
      LPRINT "STUDENT NUMBER " INT(I / N + 1)
1440  FOR H = 0 TO N - 1:
       LPRINT TAB(5)B(I + H);
1450   NEXT H:
      LPRINT :
      NEXT I
1460 GOTO 530
1470 LPRINT :
     LPRINT :
     LPRINT "FOR QUESTION "H + 1:
```

Program continued

```
     LPRINT TAB(25)"PASS"; TAB(35)"FAIL":
     LPRINT "INSTRUCTED"; TAB(25)A; TAB(35)B:
     LPRINT "NON-INSTRUCTED"; TAB(25)C; TAB(35)D:
     GOTO 1250
1480 LPRINT "PHI COEFFICIENT FOR QUESTION "H + 1" = "X
1490 IF Z$ = "P"
      THEN
       1520
1500 PRINT "DENOMINATOR FOR THIS QUESTION = "B2". NUMERATOR = "A2:
     IF A2 > 0 PRINT "QUESTION IS GOOD.":
     GOTO 1300
1510 PRINT "QUESTION IS BAD.":
     GOTO 1300
1520 LPRINT "DENOMINATOR FOR THIS QUESTION = "B2". NUMERATOR ="A2:
     IF A2 > 0 LPRINT "QUESTION IS GOOD.":
     GOTO 1300
1530 LPRINT "QUESTION IS BAD.":
     GOTO 1300
1540 INPUT "1=ADD INSTRUCTED STUDENT. 2=ADD NON-INSTRUCTED STUDENT. W
     HICH";E:
     ON E GOTO 80,160
1550 OUT 255,4
1560 INPUT ;A$
1570 END
```

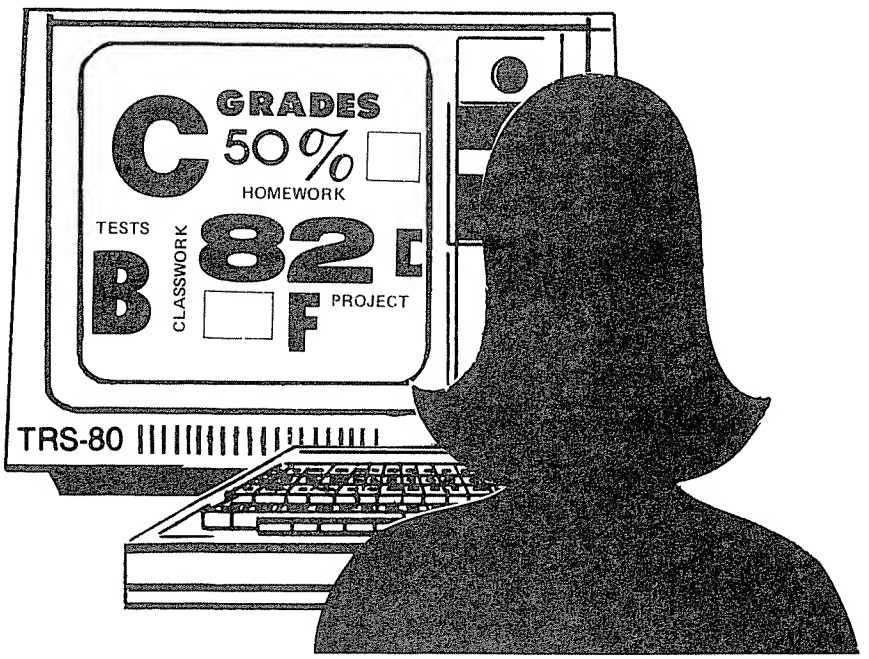
Using a TRS-80 to Tabulate Student Ratings

by Anne Weiss

Too often, while teachers concentrate on students, they forget about their colleagues. Many of them have no direct experience with computers. The tendency is to view the machines as mystifying monsters who make errors on their bills.

Our 16K Level II TRS-80, with its ability to perform arithmetic calculations quickly and to make complicated if-then decisions tirelessly, seemed a natural to help ease the burden of assigning grades at the end of each marking period. After discussing the various methods of grading with several teachers at St. Peter's High School in New Brunswick, NJ, a program called Grade Book was developed. The program was designed to be flexible enough to meet the needs of different teachers.

Programs such as Grade Book not only help to lessen the work involved in performing necessary chores, but also have nice fringe benefits. They can help dispel any computer-related anxieties and can help make teachers more receptive to using computers in their own classes. For example, we are preparing presently to have a class of sociology students use the computer



simulation Hammurabi to investigate the problems involved in ruling a country. Also, our tape library should be increasing by leaps and bounds, since some typing students will be getting practice by keying in programs from books and magazines.

Another necessary chore that must be carried out annually is the tallying of points for those students who are candidates for the National Honor Society. Every spring our NHS moderator spends a good deal of time counting the ratings given eligible students by each faculty member. Her summaries then go to a committee for final consideration. Here again, our TRS-80 is helping to get the job done.

Our administration and faculty are asked individually to rate each candidate with respect to their qualities of character, leadership, and service to the school, along with an overall recommendation for election to the Society. The ratings are on a scale of 0 (lowest) to 5. In the past, there was a dilemma about what to do when certain teachers could not rate a particular student in some or all of the categories. The question was how to differentiate between a given rating of 0 and one of no rating due to lack of knowledge of the student.

It was decided to use a method similar to that employed in Grade Book to differentiate between an assigned grade of 0 and an excused absence or omission. In the first case the student is penalized, but not in the second. In the NHS program, the user is instructed to enter 0 for a rating of zero, and −1 to indicate that a rating was not given. When the scores are tallied, the −1s are ignored.

We also wanted the summary to include the names of any teachers who gave a low recommendation rating (0, 1, or 2), as well as how many positive or negative comments were listed for each student. An appropriate form was devised to facilitate data handling (see Figure 1). Each faculty member was

PLEASE CIRCLE THE APPROPRIATE RESPONSE IN EACH CATEGORY. IF YOU DO NOT KNOW A PARTICULAR STUDENT, CIRCLE 'NO' AND GO ON TO THE NEXT STUDENT

Student ( )	Able to Rate ?	Leadership	Character	Service	Recommend For Acceptance	Comments
	Yes No	0 1 2 3 4 5	0 1 2 3 4 5	0 1 2 3 4 5	0 1 2 3 4 5	
	Yes No	0 1 2 3 4 5	0 1 2 3 4 5	0 1 2 3 4 5	0 1 2 3 4 5	
	Yes No	0 1 2 3 4 5	0 1 2 3 4 5	0 1 2 3 4 5	0 1 2 3 4 5	
	Yes No	0 1 2 3 4 5	0 1 2 3 4 5	0 1 2 3 4 5	0 1 2 3 4 5	
	Yes No	0 1 2 3 4 5	0 1 2 3 4 5	0 1 2 3 4 5	0 1 2 3 4 5	
	Yes No	0 1 2 3 4 5	0 1 2 3 4 5	0 1 2 3 4 5	0 1 2 3 4 5	
	Yes No	0 1 2 3 4 5	0 1 2 3 4 5	0 1 2 3 4 5	0 1 2 3 4 5	
	Yes No	0 1 2 3 4 5	0 1 2 3 4 5	0 1 2 3 4 5	0 1 2 3 4 5	

Figure 1

given a description of the categories and the meanings of the various responses. To hand-tally the results would be quite tedious and often would present the possibility for error. Fortunately, the TRS-80 can grind out such summaries tirelessly and accurately in practically no time. Our small Quick Printer II gives us a compact record of each student's ratings. This information is then added to the student's academic record and other credentials and is presented to the election committee as a neat package.

The program seems versatile enough to be able to be adapted for use in other tallying situations. Figure 2 shows a typical printout for a student.

```
STUDENT #1          # OF TEACHERS
LEADERSHIP    80          20
CHARACTER     60          20
SERVICE       60          15
RECOMMEND     76          19
20 TEACHERS RATED STUDENT
17 DID NOT
CUMULATIVE SCORE OF 276 POINTS
OVERALL RATING IS 15
5 POSITIVE REMARKS
2 NEGATIVE REMARKS
RECOMMENDATION OF LESS THAN 3 BY
SMITH
JONES
```

Figure 2

Notice that there was no mention of disks or fancy line printers. If ever we are fortunate enough to obtain any, our applications will be able to touch every corner of the school. For now, we are concentrating on more bake and plant sales in order to raise the funds needed to purchase a third system.

Program Listing

```
10 REM  RECORD HONOR SOCIETY RATINGS
20 :
   '  * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
30 :
   '              BY DR ANNE WEISS
40 :
   '  * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
50 CLEAR 400:
   DIM T$(40):
   REM  RESERVE SPACE FOR 40 TEACHER NAMES
60 CLS :
   DIM A$(4),A(4),K(4),X(4)
70 A$(1) = "LEADERSHIP":
   A$(2) = "CHARACTER"
80 A$(3) = "SERVICE":
   A$(4) = "RECOMMENDATION"
90 PRINT "FOLLOW THE DIRECTIONS GIVEN AND PUSH THE ENTER KEY"
100 PRINT "AFTER EACH ANSWER":
    PRINT :
    PRINT
110 PRINT "EACH STUDENT SHOULD HAVE A NUMBER, AS WELL AS EACH TEACHE
    R":
    PRINT :
    PRINT
120 PRINT "CHECK THE SCREEN AS YOU GO ALONG TO CATCH ANY ERRORS":
    PRINT :
    PRINT
130 INPUT "PUSH ENTER WHEN YOU ARE READY TO START ";Q$:
    CLS :
    GOSUB 780
140 :
    '
150 REM  INITIALIZE FOR EACH STUDENT
160 :
    '
170 FOR S = S3 TO X
180  T = 0:
     P = 0:
     N = 0:
     U = 0:
     G = 0:
     Z = 0
190  FOR I = 1 TO 4:
      A(I) = 0:
      K(I) = 0:
      NEXT
200  :
     '
210  REM  PROCESS EACH TEACHER
220  :
     '
230  FOR T1 = 1 TO Y
240   PRINT :
      PRINT "DOES TEACHER #";T1; "KNOW STUDENT #";S;" (Y OR N)":
      INPUT K$
250   F = 0:
      REM  RESET LOW RECOMMENDATION FLAG
260   IF K$ = "N"
       THEN
        U = U + 1:
        GOTO 410:
        REM  COUNT TEACHERS NOT KNOWING STUDENT
270   IF K$ = "Y"
       THEN
        290
280   PRINT "DO AGAIN PLEASE":
      GOTO 240
290   PRINT :
      PRINT :
```

```
      PRINT :
      PRINT "ENTER RATINGS FOR STUDENT #";S;"BY TEACHER #";T1:
      PRINT
300   PRINT "USE -1 WHEN QUESTION IS NOT ANSWERED"
310   PRINT "USE -2  TO REDO ANY TEACHER'S RATINGS"
320   FOR I = 1 TO 4
330    PRINT :
       PRINT A$(I);:
       INPUT X(I)
340    IF X(I) > 5 OR X(I) < - 2 OR X(I) < > INT(X(I)) PRINT "DO AGA
       IN, PLEASE":
       GOTO 330
350    IF X(I) = - 2
        THEN
         CLS :
         GOTO 240:
         REM  CORRECT AN ERROR
360    NEXT I
370   IF X(4) < 3 AND X(4) > - 1 T = T + 1:
      F = 1:
      INPUT "TEACHER'S NAME"; T$(T):
      REM  LOW RECOMMEND
380   PRINT :
      INPUT "REMARKS P=POS, N=NEG, 0=NONE";Q$
390   IF Q$ = "P" OR Q$ = "N" OR Q$ = "0"
       THEN
        410
400   PRINT "DO AGAIN PLEASE":
      GOTO 380
410   A$ = "":
      INPUT "PUSH ENTER TO PROCEED - USE 0 TO DO OVER ";A$:
      REM  FINAL CHECK
420   IF A$ < > ""
       THEN
        710
430   IF K$ = "N"
       THEN
        490:
        REM  GO ON TO NEXT TEACHER
440   IF Q$ = "P"
       THEN
        P = P + 1:
        REM  ANOTHER POSITIVE COMMENT
450   IF Q$ = "N"
       THEN
        N = N + 1:
        REM  ANOTHER NEGATIVE COMMENT
460   FOR I = 1 TO 4
470    IF X(I) > - 1
        THEN
         A(I) = A(I) + X(I):
         K(I) = K(I) + 1:
         Z = Z + X(I):
         REM  SKIP -1 RESPONSES
480    NEXT I
490   NEXT T1
500   :
510   REM  PRINT OUT STUDENT'S SUMMARY
520   LPRINT "STUDENT #";S; TAB(19);"# OF TEACHERS"
530   FOR I = 1 TO 4
540    LPRINT A$(I); TAB(13);A(I); TAB(25);K(I)
550    NEXT I
560   LPRINT T1 - U - 1;"TEACHERS RATED STUDENT"
570   IF U > 0 LPRINT U;"TEACHERS DID NOT"
580   FOR I = 1 TO 4:
       IF K(I) = 0
        THEN
         600
```

Program continued

```
590   G = G + A(I) / K(I)
600   NEXT I
610  LPRINT "CUMULATIVE SCORE OF";Z;"POINTS"
620  LPRINT "OVERALL AVERAGE RATING IS"; INT(G + .5)
630  IF P > 0 LPRINT P;"POSITIVE REMARKS"
640  IF N > 0 LPRINT N;"NEGATIVE REMARKS"
650  IF T = 0
      THEN
       680
660  LPRINT "RECOMMENDED LESS THAN 3 BY"
670  FOR I = 1 TO T:
      LPRINT T$(I):
      NEXT I
680  LPRINT :
     LPRINT :
     LPRINT :
     PRINT :
     PRINT
690  NEXT S
700 :
    '
710 REM    REDO A PARTICULAR TEACHER
720 :
    '
730 IF F = 1
     THEN
      T = T - 1
740 :
    '
750 IF K$ = "N"
     THEN
      U = U - 1
760 CLS :
    GOTO 240
770 :
    '
780 REM  GET STARTING PARAMETERS
790 :
    '
800 INPUT "HOW MANY STUDENTS ALL TOGETHER ";X:
    PRINT
810 INPUT "WHICH STUDENT # STARTING WITH ";S1:
    REM  IN CASE THIS IS A CONTINUATION
820 PRINT :
    INPUT "HOW MANY TEACHERS DID RATINGS ";Y
830 IF PEEK(14312) < > 127 PRINT "PLEASE CHECK THAT THE PRINTER IS R
    EADY TO GO":
    INPUT "PUSH ENTER WHEN PRINTER IS READY ";Q$
840 RETURN
845 :
    '
850 :
    '  A( ) = SUMS IN 4 CATEGORIES    X( ) = INPUTS OF 4 CATEGORIES
860 :
    '  K( ) = # TEACHERS RATING       A$( ) = CATEGORY NAMES
870 :
    '  T$( ) = LOW RECOMMENDERS       K$ Y OR N IF KNOW STUDENT

880 :
    '  F = LOW RECOMMEND FLAG         G = RATING AVERAGE
890 :
    '  N = # NEGATIVE COMMENTS        P = # POSITIVE COMMENTS
900 :
    '  S = STUDENT BEING RATED        S1 = STARTING STUDENT #
910 :
    '  T = # LOW RECOMMENDS           T1 = TEACHER DOING RATING
920 :
    '  U = # TEACHERS NOT KNOWING     X = # OF STUDENTS RATED
930 :
    '  Y = # OF TEACHERS RATING       Z = SUM OF ALL RATINGS
940 :
    '  Q$ = INPUT VARIABLE
```

GAMES

Swords and Sorcery II

The President Decides

Babe Ruth Is Alive and Well and Hitting Home Runs on My TRS-80

Swords and Sorcery II

by Barry L. Adams

The August 1978 issue of *Kilobaud Microcomputing* magazine carried a game program entitled Swords and Sorcery. The program was written on an SWTP 6800 machine using an 8K BASIC interpreter. The game involved you in the search and rescue of a princess held in an evil captor's dungeon. During the quest, the player encountered a variety of creatures —some good, some bad.

I immediately fell in love with the program and soon adapted it to my Level II 16K machine. As written, the listing required less than 8.5K of memory. When it comes to writing game programs, I generally follow the advice of my old art teacher: When you're painting a picture, fill up the entire canvas; after all, you're paying for it.

Armed with this bit of philosophy and a great deal of unused space in my 16K memory, I set out to paint my canvas.

New Dimensions

The revised program still follows the overall theme of the original, but adds more dimension to the encounters that pop up during play. The program uses TRS-80 graphics and includes a few new adversaries. Whereas the original program is primarily random based, the revised program also includes the elements of skill, strategy, and awareness.

The scenario of the game is familiar. In typical fairy tale fashion, you are an impoverished, rather inept hero, attempting to rescue an elfin princess who has been imprisoned in a dungeon located deep within the Old Forest. When you begin your quest, you are equipped with only a small sword and some provisions.

Prior to entering the Old Forest you may be offered assistance from a dryad, as well as the counseling of the Great Oracle. The nymph is a real plus as she makes an excellent guide and can be helpful in combat with the trolls. However, be careful not to offend her, because she can turn that magical power on you with a curse.

The Oracle, on the other hand, appears to be more interested in the maidens you're bringing rather than helping. Nevertheless, once appeased, he can point you in the right direction.

Before you start your journey, familiarize yourself with two units of measure in this little magical world: the Yerb and the Farbble Warfer. Both are measures of distance. Legend has it that the measure was defined as the distance between Ezekiel Yerb's house and that of Hansel Farbble Warfer. One man was short and the other rather tall, and as a result the

two could never agree on the number of steps between the two houses. The sense of the whole thing has long since been lost, and, today, all we know is that 1 Yerb is equal to ¾ of a Farbble Warfer.

While on your journey, you will meet a number of different critters. The full cast of characters is as follows:

Nymph: She knows the Old Forest like the back of her hand and is very good in fights with trolls.

Hot dog salesman: Ha! You thought that you could get away from them, yet here they are. Don't laugh! One of these gastronomical marvels can keep you going long after your provisions have given out.

The Great Oracle: Generally, he has more interest in what you can do for him, rather than what he can do for you. However, he may tell you the correct path to take.

Slave girls: They provide conversation and something for the satyrs to look at; otherwise, they do little but cut into your provisions.

Rats: They will give you the willies and make you run, but otherwise are of little consequence.

Snakes: A snake bite will lay you up for a day.

Spiders: They will attack, unless you can outrun them.

Dragons: Dragon slaying is still big news with big rewards, but be careful!

Goblins: You can be startled by them and run, otherwise they will enslave you, sell you to the satyrs, or let you go free for ransom.

Trolls: There are two kinds of trolls—your everyday run-of-the-mill troll and the dreaded warrior trolls. The common trolls are pesky fighters, particularly dangerous in the early going, while the warrior trolls are bad news all of the time.

The Necromancer: The chief heavy and captor of the princess, as well as the all-around bad egg.

Elements of the Fantasy

The other elements of the fantasy consist of the pits into which one occasionally tumbles, gold coins, and an enchanted sword. As you might have guessed, the pits are an obstacle from which you must escape either by climbing or yelling for help. The gold that you pick up along the way is used to buy food, pay ransom, and provide you with a little bankroll should you be lucky enough to complete your quest.

Of course, there is a "Catch 22." The weight of all that gold is somewhat of an encumbrance to combat and inhibits your fighting ability. The enchanted sword, on the other hand, enhances your fighting ability. In fact, you are usually in big trouble if you don't have it.

A player's fighting skill develops with successful combat. A player in-

creases fighting ability in combat with common trolls, satyrs, and dragons. But there are degrees of improvement. More fighting ability is acquired by slaying dragons than trolls. However, a similar gain is made when either a troll or a satyr runs from you.

In like fashion, your own fighting ability is diminished when you run from combat. Dragon fighting is the only exception. It's your choice. You can walk away from it any time.

Considerable fighting ability is usually necessary to defeat a warrior troll. But after fighting one of the super trolls, you are usually so frazzled that your fighting ability has been reduced.

The playing instructions are simple. Prior to loading the program from cassette, set the memory size to 32697 in order to reserve room for the dragon's graphics code. If you are already up and running, you can get back to MEMORY SIZE? without powering down by entering the SYSTEM command followed by /0 ENTER.

Initially, entering RUN will get the introductory title and the familiar READY. At that point the first five lines of the program are automatically deleted. Entering RUN thereafter initiates the main program. This is done to conserve memory, providing an additional 600 or so bytes.

The INKEY$ command is used to eliminate the repeated use of the ENTER key where possible.

There are questions which require word answers: RUN, YES, NO, CLIMB, and YELL. Only the first letter of the word need be typed. In two cases a prompt answer is important—encounters with either the spiders or the dragons. To avoid attack from a spider, the player must strike the R key as quickly as possible. In the case of the dragon, the rescuer is moved toward and away from the dragon by using the less than and greater than signs. Any other key will stop the rescuer in place.

Once you have rescued the princess, you no longer have the choice to fight or not fight common trolls. To make it to safety, you have to fight your way out.

The Listing

There are several lines that contain IF statements like IF F PRINT"THE NYMPH GOES MAD" (line 900). This is not a mistake, but a memory-saver, and is permissible because numerical IF statements test for a non-zero value. F, in this case, can only be a 1 or a 0.

The revised program requires nearly all of your 16K memory. To further conserve memory, no REM statements are used except for the listing title. However, the program has been written in more or less block format with a PRINT statement introducing each block. Table 1 lists those blocks. That

should be enough to start you on your search for the dungeon, the princess, and a typical fairy tale ending.

Have fun, but watch out for the onions on those hot dogs—they are murder.

LINE NUMBERS	DESCRIPTION
1-11	introductory title; set up dragon's graphics code
14-30	initialize variables; main program entry point
65-70	random number seed
85-105	timer gallery
110-115	question routines
120	setup for nymph
125-170	Great Oracle
180-195	set up of main program loop
200	main loop entry point
220-240	nymph guidance and lot casting
250-270	path choosing
280	enchanted sword
290	snake in the grass
300	check for dungeon and trolls
310	check for rats
320	check for dragons
330	check for pit
335	check for the dungeon
340-360	something is in the bushes
370	check for Necromancer and Satyrs
390-400	gold
410	slave girl
420	check for dungeon
430	check for pit
440	dead end
500	nymph's mad
530-540	travel advisory
550-556	check on provisions
570-600	captured by goblins
610-695	dungeon
700-745	run-of-the-mill troll
750-790	Satyrs
800-885	the pits
900-910	Necromancer
950	rats
960-1095	Warrior Troll
2000	something is in the bushes
2100-2120	spider
2300-2350	surprise goblin
2400-2430	the hot dog salesman
3000-4200	dateline: news story

Table 1. *Block format for Swords and Sorcery II*

Program Listing

```
 1 CLS :
   PRINT @278,"SWORDS AND SORCERY II":
   PRINT @408,"BY BARRY L. ADAMS":
   PRINT @467,"GREENVILLE , NORTH CAROLINA":
   PRINT @586,"BASED UPON A PROGRAM  WRITTEN BY BRUCE TURRIE"
 3 PRINT @656,"PUBLISHED IN THE AUGUST 1978 ISSUE":
   PRINT @735,"OF":
   PRINT @791,"KILOBAUD MAGAZINE":
   PRINT
 5 FOR X = 32767 TO 32709 STEP - 1:
    READ Z:
    POKE X,Z:
    NEXT :
   FOR X = 32708 TO 32697 STEP - 1:
    READ Z:
    POKE X,Z:
    NEXT :
   DELETE 1 - 11
10 DATA 160,190,191,180,184,191,191,191,191,191,188,144,160,186,191
   ,191,191,191,191,191,191,191,191,191,191,188,188,144,160,190
   ,191,147,175,191,191,191,191,191,191,191,191,159,131,179,191,181
   ,160,190,191,191,151,160,191,191,168,191,186,170
11 DATA 188,188,191,190,189,131,131,131,170,191,170,191
14 REM                    SWORDS AND SORCERY II
15 CLS :
   CLEAR 250:
   DEFINT L:
   XX = 458:
   GOSUB 65:
   FOR X = 1 TO 9:
    READ Y,Z:
    A(X) = Y:
    B(X) = Z:
    NEXT :
   DATA 15898,3,15961,7,16023,16,16086,17,15969,4,15907,2,15844,1,1
   5781,0,15717,0:
   R$ = " OUR BUNGLING HERO "
20 RANDOM :
   PA = 2:
   B$(1) = "CLANK ":
   B$(2) = "SLASH ":
   B$(3) = "WOOSH ":
   B$(4) = "BONG ":
   B$(5) = "CRASH ":
   B$(6) = "BING ":
   B$(7) = "CLANK ":
   S$(1) = CHR$(160) + CHR$(183) + CHR$(181) + CHR$(183) +
   CHR$(181) + STRING$(60,32) + STRING$(4,149):
   Q$ = CHR$(149):
   K$ = CHR$(132)
25 S$(2) = CHR$(176) + CHR$(144) + CHR$(176) + CHR$(144) +
   STRING$(59,32) + CHR$(162) + CHR$(135) + CHR$(151) + CHR$(151)
   + CHR$(167):
   FOR Y = 1 TO 5:
    FOR X = 1 TO 5:
     READ Z:                ,
     E$(Y) = E$(Y) + CHR$(Z):
     NEXT X,Y
30  E$(6) = CHR$(156) + " " + CHR$(156):
    E$(8) = "      ":
    E$(7) = CHR$(135) + " " + CHR$(135):
    E1$ = "-  -":
    E2$ = "    ":
    FOR X = 1 TO 7:
     READ Y:
```

Program continued

```
     E3$ = E3$ + CHR$(Y):
     NEXT :
    FOR X = 1 TO 7:
     READ Y:
     E4$ = E4$ + CHR$(Y):
     NEXT :
    GOTO 70
 65 CLS :
    PRINT @XX, CHR$(23);"SWORDS AND SORCERY II":
    PRINT :
    RETURN
 70 PA = 10:
    PRINT "ENTER A NUMBER BETWEEN 1 AND 9":
    GOSUB 115:
    A = AN:
    PA = 0:
    FOR X = 1 TO A:
     PN = RND(A + 5):
     NEXT :
    PA = 0:
    GOSUB 90:
    CLS :
    GOTO 120
 80 RETURN
 85 FOR T9 = 1 TO 50:
     NEXT :
    RETURN
 90 FOR T9 = 1 TO 100:
     NEXT :
    RETURN
 95 FOR T9 = 1 TO 300:
     NEXT :
    RETURN
100 FOR T9 = 1 TO 500:
     NEXT :
    RETURN
105 FOR T9 = 1 TO 1000:
     NEXT :
    RETURN
110 A$ = INKEY$:
    IF A$ = ""
     THEN
      110 :
     ELSE
      AN = ASC(A$):
      IF AN = 89 OR AN = 78 RETURN :
       ELSE
        110
115 A$ = INKEY$:
    IF A$ = "",115 :
     ELSE
      AN = VAL(A$):
      IF AN > PA,115 :
       ELSE
        PA = 2:
        RETURN
120 IF RND(PN) * 2 < = RND(PN) * RND(2) GOSUB 65:
    PRINT "A DRYAD HAS OFFERED TO BE YOUR":
    PRINT TAB(12);"GUIDE":
    PRINT TAB(6);" DO YOU WISH IT ?":
    GOSUB 110:
    GOSUB 90:
    IF AN = 89
     THEN
      F = 1 :
     ELSE
      IF RND(0) > .2 GOSUB 65:
      GOSUB 500:
      GOSUB 105
125 CLS :
```

```
     XX = 266:
     GOSUB 65
130  W = F:
     PRINT " THERE ARE THREE PATHS INTO THE OLD FOREST, HOWEVER , ON
     LY ONE  IS TRUE THE OTHERS ARE FOUL AND":
     PRINT TAB(4);"REEK OF MISERIES UNTOLD":
     PRINT :
     C = RND(3):
     Y = RND(3):
     IF RND(0) > .4
      THEN
       Y = C
140  PRINT "DO YOU WISH TO CONSULT THE GREAT";:
     PRINT TAB(12);"ORACLE ?":
     GOSUB 110:
     IF AN = 78,180 :
      ELSE
       CLS :
       XX = 202:
       GOSUB 65:
       PRINT :
       PRINT "AHA! TO GAIN FAVOR WITH THE FAT  ONE AND GET THE POOP
       YOU NEED    YOU MUST FIRST APPEASE HIM.":
       PRINT
150  PRINT TAB(2);"HOW MANY MAIDENS SHOULD BE":
     PRINT TAB(10);"SACRIFICED";:
     INPUT MD:
     IF RND(MD) < RND(PN) PRINT TAB(1);"OH OH THE ORACLE IS OFFENDED
     ":
     K = K - 1:
     GOTO 160
155  IF RND(0) > .4 - (MD / 10)
      THEN
       165
160  PRINT "THE SIGNS ARE UNCLEAR - YOU MUST ";:
     GOTO 170
165  PRINT :
     PRINT TAB(5);"THE ORACLE SAYS PATH";Y:
     PRINT TAB(5);"IS THE PATH OF TRUTH":
     P = 1
170  IF F AND RND(MD) > RND(PN) * RND(PN) GOSUB 500
180  PRINT :
     PA = 3:
     PRINT TAB(5);"CHOOSE PATH 1,2 OR 3":
     GOSUB 115:
     X = AN:
     L = RND(100) + 100:
     IF X < > C
      THEN
       K = K - 1:
       L = L + 50 :
      ELSE
       K = K + 1
190  K = K + .5 * RND(0) + F + P + .2 * M * SGN(5.1 - M):
     DT = L
195  CLS :
     PRINT TAB(21);"SWORDS AND SORCERY II":
     PRINT
200  PRINT :
     PRINT "YOU HAVE COME TO A FORK IN THE PATH":
     C = RND(2):
     Z = W - F:
     S = 8 - Z - G / (1 + Z):
     IF S < 4
      THEN
       S = 4
210  IF RND(3) > K
      THEN
       L = L + 1
220  L2 = 7 + RND(20):
```

Program continued

```
     LO = L2:
     IF F PRINT "PERHAPS YOU WOULD LIKE TO ASK THE NYMPH ?":
     GOSUB 110:
     IF AN = 78,250 :
      ELSE
       IF RND(0) > .5 + K / 50 PRINT "SHE DOESN'T KNOW":
       GOTO 250:
       :
        ELSE
         PRINT "SHE SAYS PATH ";C:
         GOTO 250
230  PRINT "WILL YOU CAST LOTS TO DECIDE ?":
     GOSUB 110:
     IF AN = 78
      THEN
       250 :
      ELSE
       PRINT "THE LOTS SAY YOU SHOULD TAKE PATH";:
       IF RND(0) > .5 + K / 10
        THEN
         X = 3 - C :
        ELSE
         X = C
240  PRINT X
250  PRINT "WHICH WAY DO YOU WISH TO PROCEED  PATH 1 OR 2 ?":
     GOSUB 115:
     B = 0:
     IF X = C
      THEN
       K = K + RND(0) * SGN(.5 - RND(0)):
       D = 0:
       GOTO 270
260  K = K - .2:
     D = - 1
270  L2 = L2 - S
280  I = I + 1:
     IF E = 0 AND RND(0) < .15 + .2 * F PRINT :
     PRINT "  WHAT LUCK ! YOU HAVE HAPPENED UPON ONE OF THE ENCHANTE
     D SWORDS OF THE OLD ONES":
     K = K + .2:
     E = 1:
     PRINT
290  IF RND(0) - .5 * (F + D) > .95 PRINT :
     PRINT " S N A K E !!!!!!!!!!!!!!!!!":
     PRINT :
     GOSUB 530
300  T = T + 1:
     IF RND(0) < .15
      THEN
       420 :
      ELSE
       IF RND(0) > .8 + (K + D) / 30 GOSUB 700:
       GOSUB 95
310  IF RND(0) > .95 GOSUB 950:
     GOSUB 95
320  IF RND(0) > .92 GOSUB 2200
330  IF RND(0) < .03 GOSUB 800:
     GOSUB 95
335  IF RND(0) < .4,420
340  PT = 0:
     IF RND(0) > .97 GOSUB 2000
350  IF RND(0) < .2 GOSUB 2000:
     GOSUB 95
360  IF RND(0) > .96 IF M < = 0 GOSUB 2000 :
      ELSE
       GOSUB 2400
370  IF RND(0) > .98 + (K - I / 10) / 100 GOSUB 900
380  IF W < > 0 AND RND(0) > .95 + (K + D) / 50 GOSUB 750:
     GOSUB 95
```

```
385  IF RND(0) < .1 GOSUB 570
390  IF RND(0) > .75 PRINT :
     PRINT "WHAT'S THIS BESIDE THE PATH":
     PRINT "A CHEST":
     PRINT "GOLD !":
     GC = RND(PN) * RND(PN):
     PRINT "THERE ARE ";GC;"COINS":
     G = G + GC:
     DS = DS - GC * .0001
400  IF T > 10 - 2 * F GOSUB 540
410  IF RND(0) < .05 PRINT " YOU MEET A SLAVE GIRL":
     W = W + 1:
     M = M + 1
420  IF L2 > 0,270 :
      ELSE
       L = L - LO * .85:
       IF L < 20 GOSUB 610
430  IF D < > - 1 OR RND(0) < .9,200 :
      ELSE
       IF RND(0) > .6 GOSUB 800
440  PRINT "OOPS THIS PATH IS A DEAD END":
     L2 = L2 + LO:
     L = L + LO:
     D = 0:
     GOTO 270
500  PRINT TAB(3);"THE NYMPH IS MOST PERTURBED":
     PRINT TAB(2);"SHE CURSES YOU AND DISAPPEARS":
     PRINT TAB(8);"INTO THE FOREST":
     F = 0:
     W = 0:
     K = K - 1:
     RETURN
530  PRINT "YOU HAVE BEEN HURT !"
540  PRINT "YOU MUST STOP AND REST BEFORE GOING ON.":
     IF F PRINT "THE NYMPH THINKS THAT THE DUNGEON IS LESS THAN";
     ABS(L - 20);"YERBS AWAY":
     :
      ELSE
       PRINT "YOU HAVE TRAVELED"; INT( ABS(DT - L) * .75);"FARBBLE W
       ARFERS"
550  T = 0:
     GOSUB 105:
     H = H + 1 + W - F:
     IF H < 4 - W + F,560 :
      ELSE
       IF J = 0 PRINT :
       PRINT "YOU HAVE JUST EATEN THE LAST OF YOUR FOOD":
       PRINT :
       J = 1:
       K = K - RND(0):
       :
        ELSE
         M = M + 1:
         K = K - RND(0) + .2 * F
560  IF RND(0) < 1 - (I + N) / 100 PRINT "TIME TO PUSH ON"
565  IF M < 14 RETURN :
      ELSE
       R4 = 7:
       GOTO 3000
570  PRINT :
     PRINT "YOU HAVE BEEN CAPTURED BY GOBLINS":
     IF E < > 1,600 :
      ELSE
       PRINT "THEY WANT THE SWORD THAT ONCE BELONGED TO THE OLD ONES
        -":
       PRINT "WILL YOU TRADE IT FOR YOUR FREEDOM ?":
       GOSUB 110:
       IF AN = 78,600 :
      ELSE
```

Program continued

```
        E = - .8:
        PRINT "IT IS THEN AGREED":
        GOSUB 100
580  IF PT AND RND(0) < .3
      THEN
       FOR T = 1 TO 50:
        PRINT "HA,HA,HA,HA,HA,HA,HA,HA,HA,HA,HA,HA,HA,HA,HA,HA,HA
        ,HA,HA,HA":
        NEXT :
       PRINT "THEY TOSS YOU BACK INTO THE PIT WHERE ";
590  IF M < 14 RETURN :
      ELSE
       R4 = 7:
       GOTO 3000
600  Q = RND(30):
     IF G > = Q PRINT "THE GOBLIN LORD FREES YOU FOR";Q;"GOLD COINS"
     :
     G = G - Q:
     GOTO 580:
     :
      ELSE
       IF W < = 0
        THEN
         R4 = 8:
         PRINT "YOU ARE ENSLAVED":
         GOTO 3000:
         :
        ELSE
         PRINT "YOU ARE SOLD TO THE SATYRS BY THE GOBLINS":
         GOSUB 760:
         GOTO 580
610  IF R = 0 PRINT :
     PRINT "LOOK! THERE IS THE ENTRANCE TO THE DUNGEON":
     GOSUB 105:
     PRINT "H";:
     FOR EX = 1 TO 61:
      PRINT "M";:
      GOSUB 80:
      NEXT :
     PRINT " !":
     PRINT "THERE APPEARS TO BE A GUARD":
     GOSUB 105:
     PRINT "IT'S TOO DARK TO SEE FROM HERE - MUST GET CLOSER ":
     GOSUB 105:
     GOTO 650
620  IF L > 0 RETURN :
      ELSE
       IF R = - 2
        THEN
         R4 = 9 :
        ELSE
         R4 = 10
630  GOTO 3000
650  CLS :
     DX = 80:
     FOR DY = 41 TO 18 STEP - 1:
      SET(DX,DY):
      NEXT :
     FOR DY = 17 TO 12 STEP - 1:
      DX = DX + 1:
      SET(DX,DY):
      NEXT :
     FOR DX = 86 TO 94:
      SET(DX,12):
      NEXT :
     DX = 94:
     FOR DY = 12 TO 19:
      DX = DX + 1:
      SET(DX,DY):
      NEXT :
     FOR DY = 19 TO 44:
```

```
      SET(102,DY):
      NEXT
655   DX = 70:
      FOR DY = 39 TO 1 STEP - 1:
       SET(DX,DY):
       NEXT :
      FOR DY = 2 TO 47:
       SET(120,DY):
       NEXT :
      WX = 6:
      LZ = 448:
      FOR EZ = 1 TO 3:
       GOSUB 685:
       NEXT EZ
660   X = 1:
      GOSUB 695:
      GOSUB 95:
      X = 3:
      GOSUB 695:
      GOSUB 95:
      X = 1:
      GOSUB 695:
      GOSUB 90:
      X = 4:
      GOSUB 695:
      GOSUB 95:
      X = 1:
      GOSUB 695:
      GOSUB 90:
      X = 3:
      GOSUB 695:
      GOSUB 90:
      X = 1:
      GOSUB 695:
      GOSUB 90:
      X = 8:
      GOSUB 695:
      IF LZ > 448
       THEN
        RETURN
680   VA = 448:
      VB = 462:
      FOR V3 = 1 TO 2:
       FOR LZ = VA TO VB:
        PRINT @LZ,E$(WX);:
        GOSUB 90:
        PRINT @LZ,E$(8);:
        WX = (3 - (WX - 5)) + 5:
        NEXT :
       GOSUB 660:
       VA = 462:
       VB = 476:
       NEXT V3:
      PRINT @VB,E$(4);:
      GOSUB 685:
      PRINT @VB,E$(5);:
      GOSUB 685:
      PRINT @VB,E$(5);:
      PRINT @VB,E$(4);:
      FOR X = 1 TO 3:
       PRINT @492,E1$;:
       GOSUB 100
683    PRINT @492,E2$;:
       GOSUB 90:
       NEXT :
      GOTO 690
685   PRINT @492,E1$;:
      PRINT @145,"Z";:
      FOR EX = 1 TO 15:
       PRINT "Z";:
       GOSUB 85:
```

Program continued

```
      NEXT EX:
     FOR EX = 1 TO RND(500):
      NEXT EX:
     PRINT @492,E2$;:
     GOSUB 95:
     PRINT @145, STRING$(16," ");:
     PRINT @492,E1$;:
     RETURN
690  PRINT @490,E3$;:
     GOSUB 100:
     PRINT @490, STRING$(8," ");:
     GOSUB 90:
     PRINT @490,E3$;:
     GOSUB 100:
     PRINT @490,E4$;:
     GOSUB 100:
     PRINT @VB,E$(2);:
     GOSUB 100:
     CLS :
     ON RND(4) GOSUB 700,960,960,700:
     R = 1:
     W = W + 1:
     PRINT :
     PRINT "OK,YOU'VE FOUND THE PRINCESS":
     PRINT "LET'S GET OUT OF HERE !":
     GOTO 620
695  PRINT @LZ,E$(X);:
     RETURN
700  PRINT :
     PRINT "UGH ! A TROLL !!!!       YOUR FIGHTING ABILITY IS";DS
     * 1000;"%":
     PRINT :
     IF LZ > 0
      THEN
       710 :
      ELSE
       PRINT "ARE YOU GOING TO FIGHT HIM?":
       GOSUB 110:
       IF AN = 89
        THEN
         710 :
        ELSE
         IF RND(3) = 1
          THEN
           DS = DS - .01
705  GOSUB 740:
     RETURN
710  TB = 0:
     IF E = 1 AND RND(0) < .2 + DS PRINT :
     PRINT "THE TROLL RUNS !":
     PRINT :
     DS = DS + .01:
     RETURN :
     :
      ELSE
       PRINT "THE FIGHT BEGINS . . . ";:
       GOSUB 100:
       FOR Z7 = 1 TO RND(PN) * 3
715    R4 = 3:
       PRINT B$( RND(7));:
       GOSUB 90:
      NEXT Z7:
     PRINT :
     IF RND(0) + DS + (DS * (E + F * 10)) > .5 PRINT :
     PRINT "THE TROLL HAS BEEN DEFEATED AND LIMPS OFF":
     PRINT :
     DS = DS + .0036:
     GOTO 725:
     :
      ELSE
```

```
      SF = .3:
      IF E = 1
       THEN
        SF = .5
720   IF RND(Ø) > SF + (DS * (E + F)) PRINT "OOOOF !":
      GOTO 3ØØØ:
      :
       ELSE
        PRINT :
        PRINT "YOU BOTH ARE TOO TIRED TO CONTINUE THE FIGHT !":
        PRINT :
        TB = 1:
        IF RND(3) < > 2
         THEN
          DS = DS - .ØØ2
725   IF LZ = Ø GOSUB 54Ø:
      RETURN :
      :
       ELSE
        IF TB = 1 PRINT "YOU MUST REST":
        GOSUB 1ØØ:
        FOR EX = 1 TO RND(4Ø):
         PRINT "REST ,";:
         GOSUB 85:
         NEXT :
        PRINT " - AGAIN !":
        GOSUB 85:
        GOTO 71Ø
730   IF LZ > Ø GOSUB 1ØØ:
      RETURN :
      :
       ELSE
        IF RND(Ø) + .Ø5 * E < .4 GOSUB 54Ø
735   RETURN
740   PRINT :
      PRINT "R";:
      FOR T8 = 1 TO PN * 5:
       PRINT "U";:
       GOSUB 8Ø:
       NEXT T8:
      PRINT "N !!!!!!!":
      PRINT :
      L = L + 4 * S * ( RND(Ø) - .7):
      T = T + 1:
      IF RND(Ø) > .7 GOSUB 8ØØ
745   RETURN
750   PRINT :
      PRINT "OH NO! SATYRS.":
      IF E AND RND(Ø) < .5 PRINT " HA, THEY RUN FROM YOUR SWORD":
      DS = DS + .Ø25:
      RETURN
760   PRINT "THEY WILL LET YOU GO FREE IS IF YOU WILL FIGHT THEIR CHA
      MPION - WHAT IS YOUR DECISION ?":
      GOSUB 11Ø:
      IF AN = 89 GOSUB 96Ø:
      IF RND(Ø) > .2 RETURN :
      :
       ELSE
        DS = DS - .ØØ2
770   PRINT "THE SATYRS WANT THE FEMALES OF YOUR GROUP":
      W = Ø:
      F = Ø:
      IF R = 1
       THEN
        R = - 2
780   PRINT "WILL YOU AGREE TO THESE TERMS ?":
      GOSUB 11Ø:
      IF AN = 78 PRINT "OH DID YOU MAKE THEM MAD - THEY DO YOU IN AND
       TAKE THE WOMEN":
      R4 = 6:
```

Program continued

```
     GOSUB 100:
     GOTO 3000:
     :
      ELSE
       PRINT "THEY TAKE THE WOMEM":
       IF RND(0) > .03 PRINT "THEY CURSE YOU":
       K = - 5
790  IF RND(0) > .3 RETURN :
      ELSE
       R4 = 5:
       PRINT "THE SATYRS CAN NEVER BE TRUSTED - THEY DO YOU IN ANYWA
      Y":
       GOSUB 100:
       GOTO 3000
800  PRINT :
     PRINT "YOU HAVE FALLEN INTO A DARK PIT"
810  PT = 1:
     IF RND(0) > .5 GOSUB 540
820  PRINT "YOU MUST ESCAPE":
     PRINT "DO YOU WANT TO TRY AND CLIMB OUT OR YELL FOR HELP ?"
830  A$ = "":
     A$ = INKEY$:
     IF A$ = ""
      THEN
       830 :
      ELSE
       A = ASC(A$):
       IF A = 67 OR A = 89 ZX = 0 :
        ELSE
         830
835  IF A = 67 PRINT "OK LET'S GIVE IT A TRY !":
     GOSUB 105:
     :
      ELSE
       860
840  IF RND(0) < .5 - ZX / 10 PRINT "YOU'VE MADE IT - YOU ARE OUT":
     RETURN :
     :
      ELSE
       PRINT "YOU FALL WHILE TRYING TO CLIMB !":
       PRINT :
       PRINT :
       IF RND(0) < .2 GOSUB 540
850  ZX = ZX + 1:
     IF ZX < 5
      THEN
       840 :
      ELSE
       PRINT :
       PRINT "PUFF PUFF! TOO DAMN DEEP GOT TO YELL FOR HELP!":
       GOSUB 105
860  CLS :
     SP = 540:
     FOR X3 = 1 TO RND(3):
      X6 = 1:
      GOSUB 885:
      FOR X4 = 1 TO RND(50) * 10:
       NEXT X4:
      CLS :
      GOSUB 85:
       NEXT X3:
      X6 = 3:
      GOSUB 885:
      GOSUB 100:
      X6 = 1:
      GOSUB 885:
      GOSUB 100:
      X6 = 4:
      GOSUB 885:
      GOSUB 105:
```

```
     CLS :
     GOSUB 85:
     X6 = 2:
     GOSUB 885:
     GOSUB 95:
     CLS :
     X6 = 2:
     GOSUB 885:
     PRINT @287,Q$:
     PRINT @351,K$:
     GOSUB 90
865  PRINT @722,"GEE ,  IT'S DARK DOWN HERE":
     GOSUB 100:
     CLS :
     X6 = 2:
     GOSUB 885:
     PRINT @283,"H E L P !":
     GOSUB 95:
     X6 = 5:
     GOSUB 885:
     X4 = 10:
     PRINT @X4,"*":
     FOR X3 = 1 TO 6:
      X4 = X4 - 1 + ( RND(3) - 1):
      PRINT TAB(X4);"*":
      GOSUB 85:
      NEXT X3
870  PRINT @660,"A ROPE HAS BEEN LOWERED":
     X6 = 1:
     GOSUB 885:
     GOSUB 105:
     PT = 0:
     Y = RND(4):
     PRINT @724,"YOU HAVE BEEN RESQUED BY ":
     IF Y = 1 GOSUB 700 :
      ELSE
       IF Y = 2 PRINT "OH NO !":
       PT = 1:
       GOSUB 570 :
        ELSE
         W = W + 1:
         IF F > 0 PRINT "THE NYMPH" :
          ELSE
           PRINT "AN OLD LADY"
880  PRINT "YOU MUST START OUT":
     GOSUB 100:
     GOTO 290
885  PRINT @SP,E$(X6):
     RETURN
900  PRINT :
     PRINT "IT'S THE NECROMANCER . . . . !":
     IF RND(0) > .6 GOSUB 740:
     RETURN :
     :
      ELSE
       IF F PRINT "THE NYMPH GOES MAD":
       W = W - 1:
       F = 0
905  IF R > 0 PRINT "HE TAKES THE PRINCESS":
     R = - 2
910  IF E = 1 PRINT "HE TAKES YOUR SWORD":
     E = - 1.5:
     PRINT "YOU ARE CAST INTO A DEEP PIT":
     PRINT :
     GOSUB 810:
     RETURN :
     :
      ELSE
       R4 = 11:
```

Program continued

```
      GOTO 3000
 950  PRINT :
      PRINT "UGH! RATS, MILLONS OF THEM":
      GOSUB 740:
      RETURN
 960  W3 = 2:
      H1 = 1 + DS:
      H2 = 1 + DS:
      W2 = 1 + .3 * RND(0):
      H3 = 2 + SGN(E):
      CLS :
      PRINT @458, CHR$(23);"IT'S A WARRIOR TROLL":
      R4 = 4:
      GOSUB 105:
      CLS :
      PRINT TAB(22);"THE BATTLE BEGINS"
 980  FOR Z2 = 1 TO 200:
       X = RND(7):
       NEXT Z2:
      PRINT :
      PRINT "YOU CIRCLE FOR POSITION";:
      FOR X2 = 1 TO 5:
       PRINT " . ";:
       GOSUB 90:
       NEXT X2:
      PRINT
 990  H1 = H1 - .05:
      H2 = H2 - .05:
      PRINT "HE ATTACKS WITH BOTH SWORD AND MACE SWINGING !":
      IF X = 1 PRINT "HE SLASHES WILDLY WITH HIS SWORD !":
      FOR T7 = 1 TO RND(6):
       PRINT "SLASH !";:
       GOSUB 90:
       NEXT :
      PRINT :
      GOTO 1030
1000  IF X = 2 PRINT "HE THRUST HIS SWORD STRAIGHT FOR THE BODY !":
      GOTO 1030:
      :
       ELSE
        IF X = 3 PRINT "HE ATTEMPS TO SEVER YOUR HEAD IN A SINGLE BLO
        W !":
        GOTO 1030:
        :
         ELSE
          IF X = 4 PRINT "HE TWIRLS THE MACE DIRECTLY TOWARD YOUR HEA
          D !":
          GOTO 1030
1010  IF X = 5 PRINT "HE SWINGS HIS MACE SAVAGELY AT YOUR BODY !":
      GOTO 1030:
      :
       ELSE
        IF X = 6 PRINT "HE GLANCES YOUR BLOW AND LAYS ON WITH HIS SWO
        RD !":
        GOTO 1030:
        :
         ELSE
          PRINT "HE KICKS SAND IN YOUR FACE AND SWINGS HIS SWORD TO C
          LEAVE THE   AIR AND YOUR HEAD ALONG WITH IT"
1020  PRINT "S";:
      GOSUB 80:
      PRINT "W";:
      GOSUB 80:
      FOR X = 1 TO 50:
       PRINT "O";:
       GOSUB 80:
       NEXT :
      PRINT "I";:
      GOSUB 80:
      PRINT "S";:
```

```
      GOSUB 80:
      PRINT "H !"
1030  IF RND(0) < = .5 + .3 * H2 / W2
       THEN
        1050 :
       ELSE
        PRINT "YOU'RE HIT !":
        H1 = H1 - .2:
        H2 = H2 - .2:
        GOSUB 100:
        PRINT TAB(15);"OOOOF !!":
        GOSUB 95:
        IF H1 > = .05 PRINT TAB(30);"YOU STAGGER AWAY . . . . . .":
        GOTO 980:
        :
         ELSE
          PRINT TAB(30);"YOU'RE DOWN !!!":
          GOSUB 100
1035  PRINT :
      PRINT "HE SLOWLY CLOSES FOR THE FINAL BLOW !":
      FOR X = 1 TO RND(16):
       PRINT " STEP !";:
       GOSUB 95:
       NEXT :
      PRINT :
      IF RND(0) > .1 + E / 20
       THEN
        1045
1040  PRINT :
      PRINT TAB(13);"YOU DESPERATELY MAKE A FINAL THRUST !!":
      GOSUB 105:
      IF RND(0) > .9 + DS PRINT TAB(22);"SORRY, YOU BLEW IT !":
      :
       ELSE
        PRINT TAB(28);" OOOF !":
        GOSUB 105:
        PRINT TAB(19);"YOU DID IT !  HE'S DOWN !!":
        GOSUB 105:
        RETURN
1045  PRINT :
      PRINT TAB(23);"YOU'RE FINISHED !!":
      GOSUB 105:
      GOTO 3000
1050  X = RND(6):
      IF X = 1 PRINT "YOU STOP HIS BLOW WITH YOUR SWORD AND BACK AWAY
       !!":
      GOTO 1085:
      :
       ELSE
      IF X = 2 PRINT "YOU DUCK UNDER HIS SWORD - VEER FROM HIS MACE
       AND ATTACK !":
      GOTO 1070:
      :
       ELSE
          IF X = 3 PRINT "YOU PARRY THEN ATTACK !":
          GOTO 1070
1060  IF X = 4 PRINT "YOU KICK HIM IN THE SHINS AND SCAMPER AWAY !":
      GOTO 1095:
      :
       ELSE
        IF X = 5 PRINT "YOU STOMP HIS TOES WITH YOUR BOOT !":
        GOTO 1095:
        :
         ELSE
          PRINT "YOU SLASH LEFT !";:
          IF RND(3) = 1 PRINT :
          :
           ELSE
            PRINT "YOU SLASH RIGHT !"
1065  PRINT "THEN THRUST STRAIGHT FOR HIS KNEES"
1070  FOR X3 = 1 TO H3:
       IF RND(0) < = .1 PRINT "YOU MISSED HIM !!!!":
       :
        ELSE
```

Program continued

```
          X = RND(H3):
          IF X = 1 PRINT "YOU GOT HIS LEG !":
          W2 = W2 - (DS + H2 / 5):
          W3 = W3 - (DS + H2 / 5):
          :
           ELSE
            IF X = 2 PRINT "YOU'VE SLASHED HIS ARM":
            W2 = W2 - (DS + H2 / 3):
            W3 = W3 - (DS + H2 / 5)
1075     IF X = 3 PRINT "YOU SCORE TO HIS BODY !":
         W2 = W2 - DS:
         W3 = W3 - (.05 + DS)
1080     NEXT X3
1085    IF W2 < .1
         THEN
          W2 = .1
1090    IF W3 > .05
         THEN
          980 :
         ELSE
          PRINT "HE'S DOWN !!!!!":
          PRINT "YOU'VE FINISHED HIM OFF!!":
          GOSUB 105:
          DS = DS * RND(0):
          RETURN
1095    W2 = W2 - (DS * RND(0)):
        W3 = W3 - (DS * RND(0)):
        GOTO 1085
2000    PRINT "HOLD IT !";:
        GOSUB 90:
        PRINT "   THERE'S SOMETHING MOVING BEHIND THAT BUSH !?!":
        GOSUB 90:
        R4 = RND(5):
        ON R4 GOSUB 700,2100,2200,2300,2400:
        RETURN
2100    CLS :
        PRINT @154,"GEEZE !!!!":
        GOSUB 95:
        PRINT @279,"A HUGH SPIDER !":
        PRINT @384,"QUICK !   R";:
        RN = RND(10) + PN:
        U2 = 0
2110    A$ = INKEY$:
        PRINT "U";:
        U2 = U2 + 1:
        IF U2 = RN
         THEN
          2120 :
         ELSE
          IF A$ = ""
           THEN
            2110 :
           ELSE
            IF A$ < > "R"
             THEN
              2110 :
             ELSE
              PRINT "N":
              GOSUB 90:
              CLS :
              PRINT @478,"WHEW !!":
              RETURN
2120    PRINT @347,S$(1):
        GOSUB 90:
        PRINT @347,"      ";:
        PRINT @412,S$(2);:
        GOSUB 90:
        PRINT @412,S$(1):
        PRINT @604,"SLURP !":
        GOSUB 90:
        PRINT @663,"BU";:
        FOR X = 1 TO 10:
         PRINT "R";:
```

```
      NEXT :
     PRINT "P !!";:
     GOSUB 100:
     PRINT "   HIC !":
     GOSUB 100:
     R4 = 2:
     GOTO 3000
2200 PRINT "HMMMMMMMMM . . . . . . . SURE IS WARM ?!?":
     GOSUB 100:
     CLS :
     PRINT @468, CHR$(23);"YIEPE !!!!":
     GOSUB 90:
     PRINT @524,"IT'S A DRAGON !!!!":
     GOSUB 95:
     CLS
2210 W3 = 0:
     FOR X2 = 1 TO 9:
      GOSUB 2290:
      NEXT :
     W3 = 0:
     X6 = 32709:
     X4 = 15704:
     X5 = 15708:
     GOSUB 2280:
     X4 = 15768:
     X5 = 15772:
     GOSUB 2280:
     X4 = 15835:
     X5 = 15836:
     GOSUB 2280:
     X7 = 16000 + RND(18):
     GOSUB 2270:
     PRINT @768,;
2215 X$ = INKEY$:
     IF X$ < > ""
      THEN
       Y2 = ASC(X$):
       IF Y2 = 44
        THEN
         Y2 = - 1 :
        ELSE
         IF Y2 = 46
          THEN
           Y2 = 1 :
          ELSE
           Y2 = 0
2220 GOSUB 2260:
     X7 = X7 + Y2:
     IF X7 > 16018
      THEN
       DS = DS + .045:
       PRINT :
       PRINT "YOU DID IT !  -  YOU SLAYED THE DRAGON !!!!":
       GOSUB 100:
       RETURN :
       :
      ELSE
       IF X7 > = 16000 GOSUB 2270 :
        ELSE
         2295
2240 IF RND(4) < > 1
      THEN
       2215 :
       ELSE
        F2 = 18:
        FOR F1 = 46 TO 30 STEP - 1:
         F2 = F2 + 1:
         IF POINT(F1,F2) IF E = 1
          THEN
           R7 = RND(2) :
          ELSE
           R7 = RND(3) :
        ELSE
         2250
```

Program continued

```
2245   IF R7 = 2 PRINT "SIZZLE - YOU'VE BEEN SCORCHED !";:
       DS = DS - .002:
       :
        ELSE
         PRINT "YE";:
         T6 = X7:
         FOR X7 = T6 TO 16000 STEP - 1:
          GOSUB 2270:
          PRINT "O";:
          GOSUB 2260:
          NEXT :
         PRINT "W !":
         R4 = 1:
         GOTO 3000
2250    SET(F1,F2):
        NEXT :
       GOSUB 90:
       F2 = 18:
       FOR F1 = 46 TO 30 STEP - 1:
        F2 = F2 + 1:
        RESET(F1,F2):
        NEXT :
       GOTO 2215
2260   FOR X3 = 0 TO 3:
        POKE X7 + X3,128:
        NEXT :
       POKE X7 + 65,128:
       POKE X7 + 66,128:
       RETURN
2270   POKE X7,136:
       POKE X7 + 1,174:
       POKE X7 + 2,140:
       POKE X7 + 3,45:
       POKE X7 + 65,151:
       POKE X7 + 66,149:
       RETURN
2280   FOR X3 = X4 TO X5:
        W3 = W3 + 1:
        POKE X3, PEEK(X6 - W3):
        NEXT :
       RETURN
2290   FOR Z1 = 0 TO B(X2):
        W3 = W3 + 1:
        POKE A(X2) + Z1, PEEK(32768 - W3):
        NEXT Z1:
       RETURN
2295   PRINT "COWARD !":
       RETURN
2300   C3 = 30:
       PRINT :
       PRINT "IT'S A GOBLIN ";:
       FOR EX = 1 TO 45:
        PRINT "!";:
        GOSUB 80:
        NEXT :
       GOSUB 80:
       PRINT :
       FOR EX = 1 TO 32:
        C3 = ABS(C3 + ( SGN( RND(3) - 2) * 5)):
        IF C3 > 56
         THEN
          C3 = 56
2350    PRINT TAB(C3);"R U N !!":
        NEXT :
       PRINT TAB(C3);"WHEW, SAFE !":
       C = RND(2):
       L = L + RND(PN) * C:
       DS = DS - .01 * ( RND(PN + 1) - 1):
       RETURN
2400   PRINT "GEE !":
       GOSUB 95:
       PRINT "IT'S A KID SELLING HOT DOGS !?!":
       IF J AND G > 0
        THEN
```

```
        HD = RND(G + G / 2) :
       ELSE
        HD = G + 1
2420  GOSUB 100:
      PRINT "THE KID SAYS HE'LL SELL YOU ONE OF HIS GASTRONOMIC DELIG
      HTS FOR";HD;"GOLD COINS":
      GOSUB 100:
      IF HD > G PRINT "SORRY, YOU'RE TOO POOR":
      RETURN :
      :
       ELSE
        G = G - HD
2430  M = M - RND(14):
      PRINT "GREAT !":
      RETURN
3000  GOSUB 100:
      CLS :
      PRINT "DATELINE : THE OLD FOREST":
      PRINT :
      ON R4 GOTO 3100,3200,3300,3400,3500,3600,3700,3800,3900,4000,42
      00
3100  PRINT " WOW! CAN";R$;"RUN. WHAT AN EXHIBITION OF BLINDING SPEED
      . UNFORTUNATELY IT OCCURRED AS A RESULT OF A BLISTERING DISCOVE
      RY CONCERNING DRAGONS AND IN THE OPPOSITE DIRECTION OF THAT OF
      THE PRINCESS.":
      GOTO 4500
3200  PRINT "  WHILE SEARCHING FOR THE LOST PRINCESS";R$;"BECAMETHE M
      AIN COURSE OF A RAMBLING ARACHNID":
      GOTO 4500
3300  PRINT " ";R$;"WAS ABLE TO HAVE A VERY CLOSE LOOK AT ONE OFTHOSE
       MUCH TALKED ABOUT TROLL SWORDS TODAY  -  UNFORTUNATELY IT WAS
      WHILE HE WAS BEING STABBED WITH IT":
      GOTO 4500
3400  PRINT "BLUNDER MAN STRIKES AGAIN":
      PRINT R$;"STUPEFIES EVERYONE - MAKES TROLL'S SWORD     DISAPPEA
      R IN BODY - UNFORTUNELY HIS OWN":
      GOTO 4500
3500  PRINT "AFTER REACHING FULL AGREEMENT WITH";R$;"THE SATYRS NOT O
      NLY THREW A GREAT FEAST IN HIS HONOR BUT MADE HIM THE MAIN DISH
       AS WELL":
      GOTO 4500
3600  PRINT "  WHILE";R$;"WAS CONDUCTING VERY DELICATE NEGOTIA -TIONS
       WITH THE SATYRS - THEIR DIPLOMATIC CORP ATE HIM FOR LUNCH":
      GOTO 4500
3700  PRINT "  ";R$;"FOUND THAT WHILE ON HIS LOFTY EXCURSION HE COULD
       DO WITHOUT MANY THINGS. UNFORTUNATELY FOOD WAS NOT ONE OF THEM
       - HE STARVED TO DEATH":
      GOTO 4500
3800  PRINT "  YOU GUESSED IT !";R$;"HAS DONE IT AGAIN.":
      PRINT "HOPE HE LIKES DOING WINDOWS":
      GOTO 4500
3900  PRINT R$;"MAKES IT TO DUNGEON AND BACK THROUGH MANY     PERILS -
       HAS ONLY ONE PROBLEM - LOOSES PRINCESS":
      GOTO 4500
4000  PRINT R$;"HAS PULLED IT OFF - THE PRINCESS HAS BEEN     RESQUED"
      ;:
      IF G > RND(30) PRINT ;"  - IS IMMEDIATELY ACCEPTED INTO THE KIN
      G'S COURT AND IS ALLOWED TO DO ALL THOSE NICE LITTLE THINGS THA
      T ONE DOES HAPPILYEVER AFTER":
      GOTO 4500
4100  PRINT "  -  UNFORTUNATELY HE IS TOO POOR TO BE ACCEPTED IN TO
       ROYALTY - MUST KEEP UP THE IMAGE YOU KNOW":
      GOTO 4500
4200  PRINT R$;"RAN INTO SLIGHT DIFFICULTY - THE NECROMANCER.";:
      PRINT "INFORMED SOURCES SAY THAT OUR HERO NOW EATS HEY AND IS H
      EARD TO BREY OCCASSIONALLY"
4500  PRINT :
      INPUT "ENTER FOR ANOTHER ADVENTURE";A:
      RUN
5000  DATA 156,172,32,156,172,159,175,32,159,175,140,188,32,140,188,1
      88,140,32,188,140,176,188,32,176,188,138,156,172,32,168,140,158
      ,138,140,188,32,140,188,133
```

The President Decides

by Clinton Morey

The president of the United States receives a telegram from the U.S. Ambassador to Panama. Panamanians are rioting in the streets and throwing bombs at American citizens in the Canal Zone. The president's advisors are divided on what to do. Some suggest taking a hard line, condemning Panama and ordering U.S. troops into the streets. Others urge caution, putting the troops on alert, but taking no overt action to further worsen the situation. What will the president do?

Computer Re-Creations

Each student in my high school government class had to make that decision, as if he were the president. The U.S.-Panama crisis of 1964 occurred when Lyndon Johnson was president of the United States. To give my students an understanding of the types of decisions a president must make, I designed a computer program to recreate this minor crisis in our nation's past.

The program is not a simulation but a re-creation of an historical event, with some allowance for student interaction. It's run on a Level II 16K. In this historical re-creation, the computer compares the student's responses with those made by President Johnson, but follows closely the true course of events regardless of the student's decisions. This allows the student to learn about an historical incident in a high interest situation, to compare his decisions with those of a real president, and to evaluate the quality of the decisions made.

The computer program gives information to the president in bits and pieces. Messages from the American ambassador or the senior military officer come to the desk of the president (Figure 1), requiring him to make some kind of decision. Later developments often require the president to alter his decisions as new facts are revealed. And, just as in real life, the president's advisors suggest alternate courses of action.

This bit-by-bit acquisition of information is important in helping the student understand the decision-making process. We could all be great presidents, if we had the gift of hindsight.

Classroom Discussion

There are no right or wrong answers in the course of this historical re-creation. Although some students make decisions that could possibly lead to war, the value of this computer program comes from an empathy for the tough decisions that a president must make. Thus, a very important part of

the activity takes place after each student has gone through the program, in the form of classroom discussion.

The program is not really a game—but it has proven to be fun for my students who have used it. From a teacher's standpoint, the value of the program is not in the level of enjoyment, but in the changes that occur in the student. In my classes, I have seen significant changes in students' perceptions of the presidency after working through this program.

I believe there is a place in computer instruction for historical re-creation, not just recreation.

DATE: JANUARY 9, 1964
TO: PRESIDENT MOREY
FROM: U.S EMBASSY TO PANAMA

LARGE CROWDS OF PANAMANIAN CITIZENS HAVE GATHERED ALONG THE CANAL ZONE BOUNDARY, SHOUTING, JEERING AND THROWING ROCKS AND ANYTHING THAT CAME TO HAND.
RIOTING HAS BROKEN OUT IN COLON AND PANAMA CITY.
PANAMANIAN STUDENTS AND CITIZENS HAVE THROWN MOLOTOV COCKTAILS AT BUILDINGS AND AUTOMOBILES.
CARS WITH CANAL ZONE LICENSE PLATES HAVE BEEN ATTACKED AND THEIR OCCUPANTS PULLED OUT AND SAVAGELY BEATEN.
PANAMANIAN AUTHORITIES HAVE BEEN OF LITTLE HELP. PANAMANIAN POLICE FORCES HAVE STOOD ASIDE. PANAMANIAN NATIONAL GUARD UNITS HAVE REMAINED IN THEIR BARRACKS.
CANAL ZONE POLICE FORCE TOO SMALL (ONLY 80 MEN) TO CONTROL RIOTERS. REQUEST U.S. ARMY TROOPS BE CALLED OUT TO PROTECT CANAL ZONE.

Figure 1. *Message from the Embassy*

Program Listing

```
 10 REM   THE PRESIDENT DECIDES...PART 1
 15 DIM D$(14)
 20 CLS :
    PRINT @460,"THE PRESIDENT DECIDES...PART1":
    FOR X = 1 TO 800:
     NEXT X:
    CLS :
    PRINT "YOU ARE ABOUT TO BEGIN A SIMULATION CONCERNING THE TYPES
    OF":
    PRINT "DECISIONS THAT MUST BE MADE BY AMERICAN PRESIDENTS."
 25 GOSUB 10000
 30 PRINT "THE SITUATIONS YOU ARE GOING TO FACE ARE REAL.  PRESIDENT
    S IN   THE PAST HAVE HAD TO DEAL WITH THESE ISSUES."
 35 GOSUB 10000
 40 PRINT "AS YOU STUDY THE EVENTS AND MAKE DECISIONS TRY TO EVALUAT
    E":
    PRINT "NOT ONLY YOUR CHOICES BUT ALSO THOSE OF THE PRESIDENT WHO
    ":
    PRINT "MADE THE REAL LIFE DECISIONS."
 50 GOSUB 10010
 60 INPUT "ENTER YOUR LAST NAME";N$
 70 CLS :
    PRINT "WELCOME TO THE HALLS OF POWER PRESIDENT ";N$;"."
 80 PRINT :
    PRINT "YOU ARE GOING TO GO THROUGH A TIME OF TESTING SIMILIAR TO
     THAT":
    PRINT "FACED BY PRESIDENT LYNDON JOHNSON IN 1964.":
    GOSUB 10000
 90 PRINT "THE DECISIONS YOU MAKE WILL BE COMPARED TO THE DECISIONS
    MADE BY PRESIDENT JOHNSON DURING THAT PERIOD.":
    FOR X = 1 TO 1200:
     NEXT X
100 PRINT "FOR THE SAKE OF THE ENTIRE COUNTRY, I WISH YOU THE BEST O
    F":
    PRINT "LUCK IN YOUR IMPORTANT TASK PRESIDENT ";N$
110 GOSUB 10010
130 CLS :
    PRINT "DATE:  JANUARY 7, 1964":
    PRINT :
    PRINT
140 PRINT "PRESIDENT ";N$;" YOU HAVE BEGUN SERVING YOUR NEW TERM AS"
    :
    PRINT "PRESIDENT ONLY SIX WEEKS AGO.  A MESSAGE ARRIVES AT YOUR
    DESK.":
    FOR X = 1 TO 1200:
     NEXT X
150 LPRINT "DATE: JANUARY 7, 1964"
160 LPRINT "TO: PRESIDENT ";N$
170 LPRINT "FROM: U.S. EMBASSY IN PANAMA"
180 LPRINT "-------------------------------------------------------
    ----------------------------------------------------"
190 LPRINT "A GROUP OF AMERICAN HIGH SCHOOL STUDENTS AT BALBOA HIGH
    SCHOOL IN THE PANAMA CANAL ZONE HAVE RAISED THE AMERICAN FLAG IN
    "
200 LPRINT "FRONT OF THE HIGH SCHOOL BUILDING.  THE PEOPLE OF PANAMA
     ARE AWARE OF WHAT THE STUDENTS HAVE DONE."
210 LPRINT "-------------------------------------------------------
    ----------------------------------------------------------"
220 PRINT @896,"READ THE MESSAGE AND PRESS ENTER TO CONTINUE."
230 A$ = INKEY$:
    IF A$ = ""
     THEN
      230
240 CLS :
    PRINT "ALTHOUGH ON THE SURFACE THIS SEEMS LIKE AN INOFFENSIVE AC
```

```
    T,":
    PRINT "YOU ARE AWARE THAT TROUBLE COULD RESULT.":
    FOR X = 1 TO 1200:
     NEXT X
241 PRINT "THE FLAG RAISING VIOLATED AN AGREEMENT PRESIDENT KENNEDY
    HAD":
    PRINT "MADE WITH PANAMANIAN PRESIDENT ROBERTO CHIARI IN 1962.":
    GOSUB 10000
242 PRINT "THE U.S. AND PANAMA HAVE BEEN TRYING TO REACH AN AGREEMEN
    T":
    PRINT "REGARDING CHANGES IN THE 60 YEAR OLD TREATY GOVERNING U.S
    ."
243 PRINT "CONTROL OVER THE CANAL AND THE SURROUNDING ZONE.  NO BREA
    K-":
    PRINT "THROUGHS HAD BEEN ACHIEVED IN THOSE TALKS BUT THE TWO":
    PRINT "PRESIDENTS HAD AGREED THAT THE FLAGS OF THEIR TWO COUNTRI
    ES"
244 PRINT "WOULD FLY SIDE BY SIDE.":
    GOSUB 10000
245 GOSUB 10010
250 PRINT "SINCE THE SUMMER OF 1963 THE CIA HAS BEEN WARNING YOU THA
    T":
    PRINT "YOU SHOULD EXPECT DIFFICULTIES IN PANAMA IN LATE 1963 OR
    EARLY":
    PRINT "1964.":
    GOSUB 10000
260 PRINT "THE CIA HAS SAID THAT FIDEL CASTRO, WORKING CLOSELY WITH
    THE":
    PRINT "PANAMANIAN COMMUNIST PARTY, HAS BEEN SENDING GUNS, MONEY
    AND":
    PRINT "AGENTS INTO PANAMA.
270 PRINT "THE CIA HAS SAID THAT DEMONSTRATIONS WERE LIKELY AND AN":
    PRINT "ATTEMPTED COUP AGAINST THE LEGAL GOVERNMENT WAS POSSIBLE.
    "
280 GOSUB 10000
290 PRINT "THE CIA ALSO WARNS THAT IF THAT DOES HAPPEN, THE CANAL AN
    D THE":
    PRINT "CANAL ZONE WOULD BE SPECIAL TARGETS.":
    GOSUB 10000
300 GOSUB 10010
310 PRINT "JANUARY 9, 1964":
    PRINT :
    PRINT "PRESIDENT ";N$;".  TWO DAYS HAVE PASSED SINCE THE INCIDEN
    T":
    PRINT "AND THINGS IN PANAMA HAVE APPEARED STABLE.":
    PRINT "TODAY, YOU HAVE RECEIVED THIS MESSAGE:"
315 GOSUB 10000
320 LPRINT "DATE: JANUARY 9, 1964"
330 LPRINT "TO: PRESIDENT ";N$
340 LPRINT "-------------------------------------------------------
    ------------------------------------------------"
350 LPRINT "PANAMANIAN STUDENTS HAVE ORGANIZED PROTEST MARCH.  THEY
    ENTERED THE CANAL ZONE AND WENT TO BALBOA HIGH SCHOOL."
360 LPRINT "THEY FOUGHT WITH CANAL ZONE POLICE AND AS THEY LEFT THE
    CANAL ZONE, THEY BROKE WINDOWS, BURNED AUTOMOBILES, AND CAUSED"
370 LPRINT "EXTENSIVE PROPERTY DAMAGE.  SEVERAL STUDENTS AND POLICEM
    EN WERE INJURED."
380 GOSUB 10020
390 GOSUB 10010
400 PRINT "YOU MEET WITH YOUR ADVISORS AND THEY SUGGEST THE U.S.":
    PRINT "TAKE THE FOLLOWING ACTIONS:":
    PRINT TAB(5)"(1) SEND A PROTEST TO THE PANAMANIAN GOVERNMENT.":
    PRINT TAB(5)"(2) ASK THE GOVERNMENT OF PANAMA TO HELP."
410 PRINT TAB(5)"(3) ALERT U.S. ARMY TROOPS STATIONED IN THE CANAL Z
    ONE.":
    PRINT TAB(5)"(4) ORDER U.S. ARMY TROOPS TO PROTECT THE CANAL ZON
    E."
420 PRINT "-------------------------------------------------------
    -":
    PRINT "CONSIDER THE SUGGESTIONS OF YOUR ADVISORS.  YOU WILL BE A
```

Program continued

```
    SKED":
    PRINT "MAKE A DECISION ON EACH ONE.":
    GOSUB 10010
430 PRINT "PRESIDENT ";N$;":":
    PRINT TAB(5)"WOULD YOU PROTEST WHAT HAD HAPPENED TO THE GOVERNME
    NT OF":
    PRINT TAB(5)"PANAMA (YES/NO)":
    INPUT D$(1)
440 PRINT TAB(5)"WOULD YOU ASK THE GOVERNMENT OF PANAMA FOR HELP":
    INPUT D$(2)
450 PRINT TAB(5)"WOULD YOU ALERT U.S. ARMY TROOPS STATIONED IN PANAM
    A":
    INPUT D$(3)
460 PRINT TAB(5)"WOULD YOU ORDER U.S. ARMY TROOPS TO DEFEND THE CANA
    L":
    INPUT D$(4)
470 GOSUB 10000
480 CLS :
    FOR Z = 1 TO 10:
     FOR X = 1 TO 20:
      PRINT @465,"URGENT MESSAGE":
      NEXT X:
     FOR Y = 1 TO 20:
      CLS :
      NEXT Y:
     NEXT Z
490 LPRINT "DATE: JANUARY 9, 1964"
500 LPRINT "TO: PRESIDENT ";N$
510 LPRINT "FROM: U.S. EMBASSY TO PANAMA"
520 GOSUB 10020
530 LPRINT "LARGE CROWDS OF PANAMANIAN CITIZENS HAVE GATHERED ALONG
    THE CANAL ZONE BOUNDRY, SHOUTING, JEERING AND THROWING ROCKS"
540 LPRINT "AND ANYTHING THAT CAME TO HAND.":
    LPRINT "RIOTING HAS BROKEN OUT IN COLON AND PANAMA CITY.":
    LPRINT "PANAMANIAN STUDENTS AND CIVILIANS HAVE THROWN MOLOTOV CO
    CKTAILS AT BUILDINGS AND AUTOMOBILES."
550 LPRINT "CARS WITH CANAL ZONE LICENSE PLATES HAVE BEEN ATTACKED A
    ND THEIR OCCUPANTS PULLED OUT AND SAVAGELY BEATEN."
560 LPRINT "PANAMANIAN AUTHORITIES HAVE BEEN OF LITTLE HELP.  PANAMA
    NIAN POLICE FORCE HAVE STOOD ASIDE.  PANAMANIAN NATIONAL GUARD U
    NITS":
    LPRINT "HAVE REMAINED IN THEIR BARRACKS."
570 LPRINT "CANAL ZONE POLICE FORCE TOO SMALL (ONLY 80 MEN) TO CONTR
    OL RIOTERS."
580 LPRINT "REQUEST U.S. ARMY TROOPS BE CALLED OUT TO PROTECT CANAL
    ZONE."
590 GOSUB 10020
600 PRINT "YOUR ADVISORS HAVE SEEN THE MESSAGE.  THEY SUGGEST:":
    PRINT TAB(5)"(1) YOU LODGE PROTEST WITH PANAMA GOVERNMENT.":
    PRINT TAB(5)"(2) YOU REQUEST AID FROM THE GOVERNMENT OF PANAMA."
610 PRINT TAB(5)"(3) YOU ORDER U.S. ARMY TROOPS TO STATION THEMSELVE
    S IN":
    PRINT TAB(5)"THE CANAL ZONE BUT HOLD THEIR FIRE.":
    PRINT TAB(5)"(4) YOU ORDER U.S. ARMY TROOPS TO PROTECT CANAL ZON
    E AND"
620 PRINT TAB(5)"FIRE ON PANAMANIAN IF NECESSARY.":
    PRINT "-------------------------------------------------------"
630 PRINT "DECIDE WHAT YOU WILL DO ON EACH OF THESE REQUESTS."
640 GOSUB 10010
650 PRINT "PRESIDENT ";N$;", PLEASE INDICATE YOUR DECISION ON THE FO
    LLOW-":
    PRINT "ING SUGGESTIONS.":
    GOSUB 10000
660 INPUT "LODGE A PROTEST WITH THE PANAMANIAN GOVERNMENT";D$(5):
    INPUT "REQUEST AID FROM THE GOVERNMENT OF PANAMA";D$(6):
    INPUT "ORDER US TROOPS INTO CANAL ZONE BUT NOT TO FIRE";D$(7)
670 INPUT "ORDER US TROOPS TO FIRE ON PANAMANIANS IF NECESSARY";D$(8
    )
700 GOSUB 10000
705 IF LEFT$(D$(7),1) = "N" AND LEFT$(D$(8),1) = "N" GOTO 720
```

```
710 CLS :
    IF LEFT$( D$(7),1 ) = LEFT$( D$(8),1 ) PRINT "YOUR ANSWERS ARE I
    NCONSISTENT PRESIDENT ";N$:
    PRINT "YOU HAVE ORDERED U.S. ARMY TROOPS INTO THE CANAL ZONE BUT
     CAN":
    PRINT "THEY FIRE ON PANAMANIANS?":
    GOSUB 10000:
    GOTO 650
720 CLS :
    PRINT "EVALUATION OF DECISIONS...":
    PRINT "PRESIDENT JOHNSON DID NOT PROTEST TO THE GOVERNMENT OF":
    PRINT "PANAMA.  HE DID ASK FOR THEIR AID AND HE DID ORDER U.S. A
    RMY":
    PRINT "TROOPS TO BE CALLED OUT BUT THEY HAD ORDERS NOT TO FIRE."
730 PRINT "-------------------------------------------------------
    ------"
740 IF LEFT$(D$(5),1) = "Y" PRINT "PRESIDENT ";N$;", YOUR DECISION T
    O PROTEST TO THE GOVERNMENT":
    PRINT "OF PANAMA WAS DIFFERENT THAN THAT MADE BY PRESIDENT JOHNS
    ON."
750 IF LEFT$(D$(6),1) = "N" PRINT "YOUR DECISION NOT TO ASK THE GOVE
    RNMENT OF PANAMA FOR AID IN":
    PRINT "THIS CRISES DIFFERED WITH THE DECISION MADE BY PRESIDENT"
    :
    PRINT "JOHNSON."
760 IF LEFT$(D$(8),1) = "Y" PRINT "UNLIKE YOU PRESIDENT ";N$;",PRESI
    DENT JOHNSON DID NOT":
    PRINT "WANT AMERICAN SOLDIERS TO FIRE ON PANAMANIAN CIVILIANS."
770 PRINT "-------------------------------------------------------
    --":
    GOSUB 10010
780 PRINT "CONTINUING WITH OUR SIMULATION WE WILL FOLLOW EVENTS AS T
    HEY":
    PRINT "CONFRONTED PRESIDENT JOHNSON BASED ON HIS DECISIONS.":
    GOSUB 10000
790 CLS :
    FOR Z = 1 TO 10:
     FOR X = 1 TO 20:
      PRINT @465,"URGENT MESSAGE":
     NEXT X:
     FOR Y = 1 TO 20:
      CLS :
     NEXT Y:
    NEXT Z
795 LPRINT "DATE: JANUARY 9, 1964"
800 LPRINT "TO: PRESIDENT ";N$
810 LPRINT "FROM: COMMANDER OF AMERICAN TROOPS STATIONED IN THE CANA
    L ZONE"
820 GOSUB 10020
830 LPRINT "AMERICAN SOLDIERS ARE BEING SHOT AT BY SNIPERS.  SEVERAL
     CASUALITES REPORTED.  REQUEST PERMISSION TO FIRE ON SNIPERS."
840 GOSUB 10020
850 CLS :
    PRINT "YOUR ADVISORS--THE SECRETARY OF STATE, THE SECRETARY OF D
    EFENSE":
    PRINT "THE DIRECTOR OF THE CIA, AND OTHER AREA SPECIALISTS ALL A
    GREE":
    PRINT "YOU SHOULD ALLOW U.S. TROOPS TO PROTECT THEMSELVES."
860 GOSUB 10000
870 PRINT "-----------------------------------------------":
    PRINT "PRESIDENT ";N$;", IT'S YOUR DECISION.  WILL YOU ALLOW U.S
    . TROOPS TO  RETURN FIRE IN THEIR EFFORTS TO PROTECT AMERICANS?"
    :
    INPUT D$(9)
875 IF LEFT$(D$(9),1) = "N" PRINT "-----------------------------------
    ---------------------------":
    FOR X = 1 TO 2000:
     NEXT X:
    GOTO 920
880 GOTO 1000
```

Program continued

```
 900 GOSUB 10000
 910 PRINT "ALTHOUGH PRESIDENT JOHNSON DID NOT CHOOSE TO AUTHORIZE U.
     S.":
     PRINT "TROOPS TO FIRE AT PANAMANIANS AS YOU DID, EVENTS SOON FOR
     CED":
     PRINT "HIM TO TAKE THAT STEP.":
     GOSUB 10000
 920 PRINT "SNIPER FIRE FROM PANAMANIANS KILLED FOUR AMERICAN SOLDIER
     S AND":
     PRINT "WOUNDED SEVERAL.  PRESIDENT JOHNSON FELT THIS REQUIRED AM
     ERICAN":
     PRINT "TROOPS TO RETURN FIRE."
 930 GOSUB 10010
1000 CLS :
     PRINT "WHILE YOU CONTINUE YOUR DUTIES ON JANUARY 9, YOUR ADVISOR
     S":
     PRINT "RECEIVE A MESSAGE FROM YOUR EMBASSY IN PANAMA AND ASK YOU
      TO":
     PRINT "RETURN TO HANDLE SOME MORE DECISIONS.":
     GOSUB 10000
1010 LPRINT "TO: PRESIDENT "N$
1020 LPRINT "FROM: U.S. EMBASSY IN PANAMA"
1030 GOSUB 10020
1040 LPRINT "PRESIDENT CHIARI HAS INDICATED TO US THAT HE WILL BREAK
     DIPLOMATIC RELATIONS WITH THE UNITED STATES BECAUSE OF THE AGRES
     SION"
1050 LPRINT "OF THE U.S. ARMY TROOPS AGAINST PANAMANIAN CITIZENS. I T
     RIED TO MAKE CLEAR TO PRESIDENT CHIARI THAT WE WERE ONLY DEFENDI
     NG"
1060 LPRINT "OUR NATIONALS AND PROTECTING TERRITORY LEGALLY UNDER OUR
      CONTROL.":
     LPRINT "PRESIDENT CHIARI SAID THE U.S. WOULD RECEIVE FORMAL NOTI
     CE OF THE BREAKING OF DIPLOMATIC RELATIONS TOMORROW."
1070 GOSUB 10020
1080 GOSUB 10000
1090 PRINT "WITH THIS NEWS, YOU GO TO BED.  YOUR ADVISORS WILL WAKE Y
     OU IF":
     PRINT "IMPORTANT NEWS ARRIVES DURING THE NIGHT.":
     PRINT :
     PRINT :
     PRINT "SLEEP TIGHT PRESIDENT ";N$;".":
     GOSUB 10000
1100 GOSUB 10000
1110 GOSUB 10010
1120 CLS :
     PRINT "JANUARY 10, 1964":
     PRINT :
     PRINT "THIS MORNING YOU MEET IN THE CABINET ROOM WITH YOUR ADVIS
     ORS":
     PRINT "TO DETERMINE WHAT YOU SHOULD DO NEXT.":
     PRINT
1130 PRINT "CIA DIRECTOR MC CONE POINTS OUT THAT TROUBLE HAS BEEN BRE
     WING":
     PRINT "IN PANAMA FOR AT LEAST 6 MONTHS.  HE SAYS PANAMA'S IRRITA
     TION":
     PRINT "OVER THE FLAG INCIDENT IS UNDERSTANDABLE, BUT THAT THE"
1140 PRINT "ACTIVITIES WHICH HAVE OCCURED SINCE THAT INCIDENT ARE PAR
     T":
     PRINT "OF A WELL-PLANNED ANTI-AMERICAN DEMONSTRATION.":
     PRINT "REVIEW THE REPORTS OF THE PREVIOUS DAY AND PRESS <ENTER>
     WHEN YOU WISH TO CONTINUE.":
     INPUT A
1150 CLS :
     PRINT "YOUR ADVISORS RECOMMEND YOU TALK DIRECTLY WITH PRESIDENT"
     :
     PRINT "CHIARI OF PANAMA.":
     INPUT "WILL YOU ASK YOUR STAFF TO PLACE A CALL TO CHIARI (YES/NO
     )";D$(10)
1160 IF LEFT$(D$(10),1) = "Y" GOTO 1200
1165 GOSUB 10000
```

```
1170 PRINT "UNLIKE YOU PRESIDENT ";N$;", PRESIDENT JOHNSON PLACED THE
     ":
     PRINT "CALL TO CHIARI.  I'M NOT SURE I AGREE WITH YOUR DECISION
     NOT":
     PRINT "TO TALK DIRECTLY WITH THE PANAMANIAN PRESIDENT."
1175 GOSUB 10000
1180 PRINT "BUT LET'S GO ON WITH THE PHONE CALL AS PLACED BY PRESIDEN
     T":
     PRINT "JOHNSON."
1185 GOSUB 10010
1200 CLS :
     GOSUB 10000
1210 PRINT "FOR A TRANSCRIPT OF THE PHONE CALL SEE THE TELETYPE."
1220 GOSUB 10000
1230 LPRINT "TRANSCRIPT OF TELEPHONE CONVERSATION BETWEEN PRESIDENT L
     YNDON JOHNSON AND PRESIDENT CHIARI.":
     LPRINT "JANUARY 10, 1964"
1240 GOSUB 10020
1250 LPRINT TAB(10)"JOHNSON:  HELLO, MR. PRESIDENT.  MR. PRESIDENT i
     WANTED TO SAY TO YOU THAT WE DEEPLY REGRET THE SITUATION OF VIOL
     ENCE"
1260 LPRINT "THAT HAS DEVELOPED IN PANAMA.  WE APPRECIATE VERY MUCH Y
     OUR CALL TO THE PANAMANIAN PEOPLE TO REMAIN CALM.  WE RECOGNIZE
     THAT"
1270 LPRINT "YOU AND I SHOULD DO EVERYTHING WE CAN TO RESTORE QUIET,
     AND I HOPE THAT YOU'LL DO EVERYTHING POSSIBLE TO QUIETEN THE SIT
     UATION"
1280 LPRINT "AND I WILL DO THE SAME.  YOU AND I SHOULD BE AWARE OF TH
     E POSSIBILITY, AND THE LIKLIHOOD, THAT THERE ARE ELEMENTS UNFRIE
     NDLY"
1290 LPRINT "TO BOTH OF US WHO WILL EXPLOIT THIS SITUATION."
1300 LPRINT TAB(10)"CHIARI:  I FEEL, MR PRESIDENT, THAT WHAT WE NEED
     IS A COMPLETE REVISION OF ALL TREATIES WHICH AFFECT PANAMA-U.S.
     RELATIONS"
1310 LPRINT "BECAUSE THAT WHICH WE HAVE AT THE PRESENT TIME IS NOTHIN
     G BUT A SOURCE OF DISSATISFACTION WHICH HAS RECENTLY, OR JUST NO
     W"
1320 LPRINT "EXPLODED INTO VIOLENCE WHICH WE ARE NOW WITNESSING."
1330 LPRINT TAB(10)"JOHNSON:  MR. PRESIDENT, I AM SENDING TOM MANN, O
     UR ASSISTANT SECRETARY OF STATE, TO YOUR COUNTRY AS MY PERSONAL"
1340 LPRINT "REPRESENTATIVE.  HE AND HIS GROUP WILL DO EVERYTHING IN
     THEIR POWER TO FIND A SOLUTION TO THE CURRENT PROBLEMS."
1350 LPRINT TAB(10)"CHIARI:  I CAME TO WASHINGTON IN 1961 AND TALKED
     WITH PRESIDENT KENNEDY ABOUT TREATY REVISIONS.  IN THREE YEARS,
     MR."
1360 LPRINT "PRESIDENT, NOT A THING HAS BEEN DONE TO ALLEVIATE THE SI
     TUATION."
1370 LPRINT TAB(10)"JOHNSON:  PRESIDENT CHIARI, WE MUST LOOK FORWARD
     AND NOT BACKWARD.  VIOLENCE IS NO WAY TO SETTLE GRIEVANCES. FIRS
     T, LET"
1380 LPRINT "US END THE VIOLENCE; THEN WE CAN BEGIN TO TALK OVER OUR
     DIFFERENCES AND FIND SOLUTIONS."
1390 LPRINT TAB(10)"CHIARI:  YOUR COUNTRY HAS OFTEN SHOWN INDIFFERENC
     E TO PANAMA'S PROBLEMS."
1400 LPRINT TAB(10)"JOHNSON:  OUR DELEGATION WILL BE ON A PLANE IN 30
      MINUTES AND WILL ARRIVE IN PANAMA IN 5 HOURS.  I CANNOT ACT MUC
     H "
1410 LPRINT "FASTER THAN THAT, MR. PRESIDENT."
1420 LPRINT TAB(10)"CHIARI:  I AM GRATEFUL FOR YOUR COOPERATION, MR.
     PRESIDENT.  I AM GLAD YOU ARE A MAN OF ACTION AND OF FEW WORDS."
1430 LPRINT "I AM SURE OUR DIFFICULTIES WILL BE IRONED OUT."
1440 GOSUB 10020
1450 GOSUB 10010
1460 PRINT "AS YOUR DELEGATION GETS READY TO LEAVE, YOUR ADVISORS SUG
     GEST":
     PRINT "YOU HAVE THEM...":
     PRINT "-----------------------------------------------":
     PRINT "(1) AGREE TO EXPLORE NEW TREATY ARRANGEMENTS IF GOVERNMEN
     T OF"
1470 PRINT "PANAMA STOPS THE VIOLENCE.":
```

Program continued

```
     PRINT "(2) TELL PRESIDENT CHIARI HIS GOVERNMENT MUST RESTORE ORD
     ER,":
     PRINT "RESUME DIPLOMATIC RECOGNITION AND AGREE TO A PLAN OF COOP
     ERA-":
     PRINT "ION IN STUDYING THE PROBLEM WITH NO PRIOR COMMITMENTS."
1480 PRINT "(3) TELL PRESIDENT CHIARI THE US DOES NOT RESPOND TO BLAC
     KMAIL":
     PRINT "AND WE WILL SEND IN ADDITIONAL TROOPS IF NEEDED.":
     PRINT "(4) AGREE TO ANY REASONABLE REQUESTS PANAMA WANTS IF THEY
      CAN":
     PRINT "RESTORE PEACE."
1485 PRINT "-------------":
     INPUT "ENTER THE NUMBER OF YOUR ORDER TO THE U.S. DELEGATION";N
1490 IF N < 1 GOTO 1450
1495 IF N > 5 GOTO 1450
1500 ON N GOTO 1600,1700,1800,1900
1600 CLS :
     PRINT "PRESIDENT ";N$;", YOUR DECISION DIFFERED FROM THAT OF PRE
     S.":
     PRINT "JOHNSON.  PRESIDENT JOHNSON DECIDED IT WOULD NOT BE WISE
     TO":
     PRINT "GIVE IN TO DEMANDS MADE BY RIOTERS.":
     GOTO 2000
1700 CLS :
     PRINT "PRESIDENT ";N$;", YOUR DECISION WAS THE SAME AS PRESIDENT
     ":
     PRINT "JOHNSON'S.":
     GOTO 2000
1800 CLS :
     PRINT "THIS IS A RATHER SERIOUS THREAT YOU ARE MAKING PRESIDENT"
     :
     PRINT N$;".  PRESIDENT JOHNSON DID NOT CHOOSE TO MAKE THIS THREA
     T.":
     PRINT "EVEN HAD IT BEEN SUCCESSFUL, THE U.S. WOULD HAVE LOST MAN
     Y"
1810 PRINT "FRIENDS IN LATIN AMERICA."
1820 GOSUB 10010
1830 GOTO 2000
1900 CLS :
     PRINT "PRESIDENT ";N$;", HOW COULD YOU!  WILL YOU GIVE IN TO ANY
     ":
     PRINT "TERRORIST OR ANY GROUP OF PEOPLE WHO WANT TO PUSH THE U.S
     .":
     PRINT "AROUND FOR THEIR OWN ADVANTAGE?"
2000 GOSUB 10010
2010 PRINT "PRESIDENT JOHNSON LET THE GOVERNMENT OF PANAMA KNOW THE U
     .S.":
     PRINT "WOULD NOT CONSIDER REQUESTS FOR TREATY NEGOTIATIONS":
     PRINT "UNTIL THE VIOLENCE HAD STOPPED AND PANAMA RESUMED DIPLOMA
     TIC":
     PRINT "RECOGNITION OF THE U.S."
2020 GOSUB 10000
2030 PRINT "IT TOOK SEVERAL DAYS BEFORE PANAMA REALIZED THE U.S. WOUL
     D":
     PRINT "NOT BACK DOWN FROM ITS STAND.":
     GOSUB 10000
2040 PRINT "PRESIDENT CHIARI, REALIZING NOTHING WAS TO BE GAINED FROM
     ":
     PRINT "FURTHER RIOTING ORDERED THE PANAMA NATIONAL GUARD OUT OF
     ITS":
     PRINT "BARRACKS.  ORDER WAS QUICKLY RESTORED.":
     GOSUB 10000
2050 INPUT "PRESS ENTER TO CONTINUE";O:
     CLS :
     PRINT "THE CRISES WAS OVER.  WITHING A FEW MONTHS PANAMA AND THE
     ":
     PRINT "U.S. WERE AGAIN  ON FRIENDLY TERMS.":
     GOSUB 10000
3000 PRINT "END OF SIMULATION."
9999 END
```

```
10000 FOR X = 1 TO 1200:
       NEXT X:
      RETURN
10010 PRINT @896,"PRESS <ENTER> TO CONTINUE."
10015 A$ = INKEY$:
      IF A$ = ""
       THEN
        10015
10016 CLS
10017 RETURN
10020 LPRINT "-------------------------------------------------------
      ------------------------------------------------------"
10025 RETURN
```

Babe Ruth Is Alive and Well and Hitting Home Runs on My TRS-80

by Ralph Hickok

The American League All-Stars seemed to have the game well in hand. They were leading, 4-0, in the bottom of the ninth, thanks to home runs by Babe Ruth and Mickey Cochrane, while Cy Young had given up only three National League hits.

Then Frank Frisch doubled and was singled home by Zack Wheat. After Bill Terry singled, Hank Aaron hit a home run to tie the score. Young retired the next two hitters, only to give up a single to Mel Ott and a game-winning double to Ernie Banks, as the National League pulled it out, 5-4.

If you know something about baseball, you're wondering when Hank Aaron ever hit a home run off Cy Young, who retired from the game 22 years before Aaron was born. Or when Zack Wheat and Ernie Banks ever could have played together.

The answer is that it all happened right in my living room, just a few days ago, on the CRT of my Level I 4K TRS-80, thanks to a program which I call Historic Baseball. The listing was converted to Level II 16K.

The game described above was one of a long series between the all time all-stars of the American and National Leagues, as chosen in a vote of sports-writers and fans in 1969, professional baseball's centennial year. With Historic Baseball, you can play games between *any* two teams of the past—or of the present, for that matter.

I wrote the original program in Level I BASIC more than a year before I had a computer, and I assumed it would require at least 16K. The original would have, but, having finally bought my 4K model, I condensed the program to fit, thereby eliminating a number of features.

Although I regret the loss of those features, I was pleasantly surprised at how much I managed to squeeze into the abridged version. Although the listing has been converted for 16K Level II, it can be used on Level I 4K if condensed according to Appendix A.

Historic Baseball is essentially an operating program that brings the hitters to the plate in the proper order, generates random numbers (based on the players' actual statistics) to determine what happens, gives a pitch-by-pitch printout on the action, and stops when the game is over—whether in nine innings or extra innings.

The hitting statistics are embedded in 18 DATA lines (1505-1545 for one team, 1600-1640 for the other). See Table 1. Pitchers' statistics are carried in two program lines (55 and 60).

5-10: Initialize and ask which team bats first
20-45: Determine which hitter is up and fetch DATA
50-65: Modify hitter's data with pitching factors
70-100: Deliver pitch—generate random number and determine result
105-325: Single—move runners and score runs, as necessary
500-525: Home run; how many runs score? Set R to 0
600-615: Triple; counts runs, if any—set R to 4
700-735: Double; move runners and score runs, as necessary
800-810: Strike—is it a foul? Increment S—is it Strike 3?
900-950: Ball—increment B—is it Ball 4?
960-990: Out—double play possibility? Go to Out subroutine
1505-1545: DATA lines; hitting statistics for Team 1
1600-1640: DATA lines; hitting statistics for Team 2
3220: Out subroutine—is side retired? If 9th inning or later, is game over?
5000: Prints top of scoreboard—inning and team at bat
5014: Prints middle part of scoreboard
5065: Prints lower part of scoreboard
7000: Change score if runs came in; does score end game?
9000: Time delay

Table 1. *Line Descriptions*

If you're playing a long series between two teams, you can have a different set of pitchers for each game simply by changing the two program lines. To insert a new set of teams, you change the 18 DATA lines, the two pitching program lines, and three lines in the scoreboard subroutine.

Using a separate data development program, I have worked out the DATA lines and pitching program lines for all of the players listed in *Daguerrotypes*, a *Sporting News* publication that includes complete year-by-year statistics on all players in the Baseball Hall of Fame and many other stars of the past. I'm now working on the data for all teams that have won pennants in either league since 1901, the beginning of baseball's modern era.

If you have access to the Macmillan *Encyclopedia of Baseball*, or a similar reference work, you can do it yourself, for any players you like.

Looking at the DATA lines before running through the rest of the program, each line contains a string variable, the hitter's position and name (e.g., 2B—CHARLIE GEHRINGER, the leadoff man for the American League All-Stars), followed by six numbers. A glance at the READ statement in line 45 reveals that the variables are designated H, J, K, L, M, and N (Table 2).

H represents the number of hits per 500 official times at bat, *not* including strikeouts. To take a simple example, if a player batted .333 on 200 hits in 600 times at bat and struck out 100 times, the number of strikeouts is sub-

tracted from the times at bat. He now has 200 hits in 500 times at bat, so his H figure is 200. This will give him a batting average of .400 for those times at bat when he doesn't strike out.

J and K represent balls and strikes, respectively. To work out these figures, I use a general formula that's plugged into the data development program. Multiply a player's total walks by 6 and total strikeouts by 4.5; then divide each figure by his *actual* times at bat, *including* walks and strikeouts; then multiply by 1100. This seems to work well, statistically.

L, M, and N are fractional figures representing the player's proportion of doubles, triples, and home runs, respectively, to total hits. In every 1,000 hits, Charlie Gehringer will have 202 doubles, 51 triples, and 65 home runs.

Now to the pitching program lines. Line 55 in this program represents

A—Indicator of which team is batting (= 1 or 2)
B—# of balls on current hitter (= 0 to 4)
C—Which player is at bat when A = 1
D—Which player is at bat when A = 2
G—# of outs per 500 at-bats for each hitter
H—# of hits per 500 at-bats for each hitter
I—Inning—incremented by .5 after each half inning; if a non-integer, it's the bottom half of the inning
J—Proportion of balls thrown to specific hitter
K—Proportion of strikes thrown to specific hitter
L—Proportion of doubles to hits
M—Proportion of triples to hits
N—Proportion of home runs to hits
O—# of outs in this half inning
R—Indicator of which bases are occupied:
= 0, bases empty
= 1, runner on first
= 2, runner on second
= 3, runners on first and second
= 4, runner on third
= 5, runners on first and third
= 6, runners on second and third
= 7, bases loaded
S—# of strikes on current hitter
V—Counter of runs scored on preceding play
W—Counter to READ current hitter's statistics; = C when A = 1, = D + 9 when A = 2; also used as a random number in some situations
X—Total runs for first team to bat
Y—Total runs for second team to bat
Z—Random number to determine result of pitch

Table 2. *Table of variables for Historic Baseball*

Walter Johnson's career statistics, and line 60 represents Christy Mathewson's. (Some time in the future, I'm going to play a similar series, based on statistics from each player's best *season*.) Each line contains three factors that change the hitter's raw H, J, and K figures. These factors are based on the pitcher's performance as compared to the average pitcher during the period—either a season or a career.

Walter Johnson's figures show that he gave up only 90.2 percent the number of hits and 74.7 percent the number of walks of the average pitcher, while striking out hitters 162 percent more frequently.

Returning to the beginning of the program to see how it runs, note the variables that are initialized: O is the number of outs; I is the inning indicator; C and D indicate which hitters are due to bat for each team and are initialized at 0 because they'll be incremented by 1 when the teams take their turns at bat; R indicates which bases are occupied and is set at 0 when there are no runners; X and Y are score indicators; and V is a run counter.

The player is asked which team bats first, and the input is A. A is used throughout as an indicator of which team is batting. In general, if A = 1, the team represented in DATA lines 1505 through 1545 is at bat; in this specific program, that's the American League.

This brings us to line 20, which is where the program goes every time there's a new hitter. This RESTOREs the data (we'll see why in a moment) and checks to see if any runs were scored on the previous play (is V greater than 0?); if so, it goes to the score-incrementing subroutine at line 7000 and then resets V to 0.

After going to subroutine 5000 to print the top part of the scoreboard, the program checks to see which team is at bat and then goes to the appropriate line to determine which hitter is due at the plate—C if A = 1, D if A = 2. After the ninth player in the order has had his turn at bat, the variable C or D becomes 10 and is then reset to 1, to bring the leadoff man up again.

W is set equal to C or to D + 9. To see why, suppose the National League is batting first. D = 1, meaning their leadoff man, Pie Traynor, is due to hit. W then equals 10. The FOR-NEXT loop at line 45 performs nine dummy READs, carrying it all the way through the American League DATA lines, and then retrieves the data for Traynor from line 1600. This explains why line 20 RESTOREs the data; the READ cycle must always begin at line 1505.

Following Traynor through his turn at bat should make the rest of the program fairly clear. Line 50 initializes the ball and strike counters, B and S, to 0. Since A = 2, the program ignores the conditional branch and drops to line 55, where Traynor's raw figures are changed by Walter Johnson's pitching factors.

The original figures were: H = 165, J = 388, K = 171. Line 55 changes them to: H = 148, J = 289, K = 277.

Line 65 is a pitcher tiring factor. The pitcher will be somewhat above average for the first three innings, exactly average (his average, that is) in the fourth, and from then on he'll become progressively less effective. Since this is the first inning, the H figure is lowered to 142.

Now that the final H figure for this time at bat has been computed, it is subtracted from 500 to find G, the out figure, and it is multiplied by L, M, and N to establish the proper figures for doubles, triples, and home runs. For Traynor, they are 21, 9, and 3.

At line 70, we go to another section of the scoreboard subroutine to print the ball and strike count and the number of outs. This is where the program comes every time there is another pitch to a hitter who's already been brought up to the plate. The CRT announces, HERE COMES THE PITCH, and a FOR-NEXT loop at line 9000 introduces a time delay.

The random number, Z, will be in the range between 1 and 1059 (500 + J + K), inclusive, in this case. If Z is less than or equal to 277 (K), the pitch is a strike, and the program branches to line 800. If it is greater than 277, but less than or equal to 566 (J + K), it's a ball—line 900. If greater than 566, but less than or equal to 924 (G + J + K), it's an out—line 960—and so on through the hits. If it fails the tests for extra base hits, control drops to line 105, and the hit is a single.

If there is a hit, the program has to determine whether any runs score and which bases are now occupied. Triples (line 600) and home runs (line 700) are relatively easy to handle. After a home run, the bases are always empty, so the R figure is always 0; after a triple, there's always a runner on third, so R = 4. All that has to be determined is how many runs score.

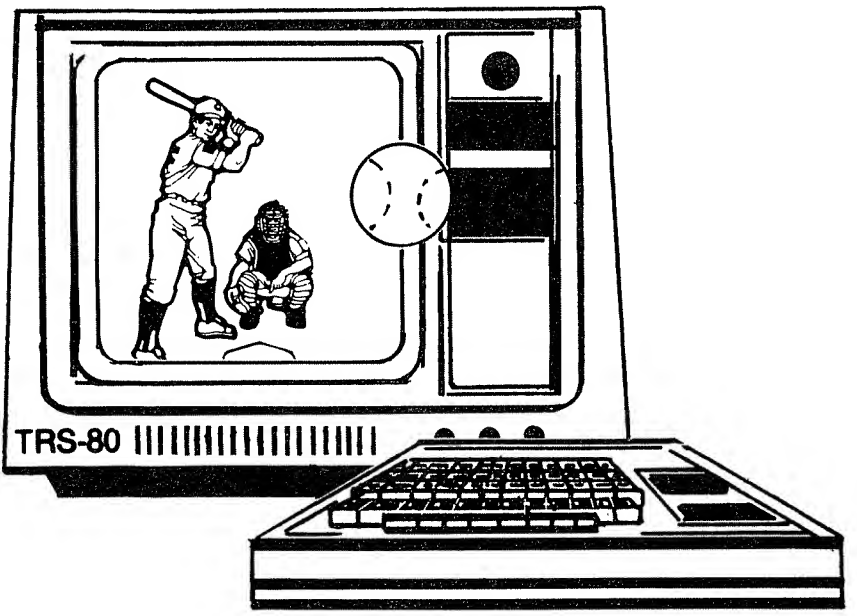
Singles and doubles are a bit more complicated. There's the possibility of a runner scoring from second on a single or from first on a double. This possibility is handled with a second random number, W. The runner will score 60 percent of the time.

The random number W is also used on a strike, to determine whether or not it's a foul ball. A foul ball will result 15 percent of the time. If there are two strikes on the hitter, a foul doesn't count as the third strike, so the program prints THE COUNT HOLDS and returns to line 70. At line 804, the strike figure is incremented and, if this is the third strike, the program goes to the out subroutine at line 3220. If not, it goes back to line 70 to update the count on the scoreboard and to deliver the next pitch.

A similar procedure is followed at 900, if the pitch is a ball. If it's ball 4, the R figure is changed; if not, it's back to line 70. For convenience, I grouped all possible R outcomes at this part of the program, lines 920-945, since I needed four of them in the case of a walk, anyway.

At line 960, where outs (except for strikeouts) come, there's a check to see whether there are less than two outs and whether first base is occupied, for

the possibility of a double play. (This line explains why I assigned R values in a seemingly arbitrary way: Any time R is an odd number, there's a runner on first, so checking is quite easy.) If the double play possibility exists, another random number is used; there will be a double play 15 percent of the time, when W is not less than 86.



The only time a runner can score on an out (unfortunately) is when there's a double play that doesn't retire the side, with a runner on third. Lines 986 through 990 deal with that situation.

Beyond the DATA lines is the out subroutine, line 3220. This increments the out counter and, if there are three outs, it changes the A figure to bring the other team to bat. It also checks to see whether this is the ninth inning or later. If so, it goes to line 3260 to determine whether or not the game is over. If the home team is leading after 8½ innings, or either team is leading at the end of Inning 9 or later, the game ends. If the game is to continue, line 3245 changes the top of the scoreboard to show the new inning and the team at bat.

The scoreboard subroutines begin at line 5000. This was originally all one subroutine, but the continual blinking of the entire scoreboard, being unnecessarily updated after each pitch, was making me seasick, so I broke it up into three segments. Lines 5005, 5008, and 5010 are the lines that have to be changed when you change teams.

The score-changing subroutine is at line 7000. Again, there's a check to

determine whether this is the game-winning run, in the bottom of the ninth or an inning later than the ninth. The last line, 7045, updates the relevant part of the scoreboard.

And, finally, line 9000 is a time delay which is called at several points in the program.

The 4K version is not interactive. With a little tinkering, however, it is possible to use a pinch-hitter or a relief pitcher. It is not terribly complicated, but you have to be a bit careful to avoid messing up the scoreboard. Here is how it's done:

Suppose our old friend Pie Traynor is due to bat. You really need a home run, and you've got Henry Aaron on the bench beside you. Aaron is approximately 7½ times as likely to hit a home run as Traynor.

You hit the BREAK key and LIST the data lines—Traynor is at line 1600. You type in a new line 1600: D.PH-HANK AARON, 176, 586, 503, .173, .032, .183. But, if you go back to line 20 at this point, that spot in the batting order will be skipped, and the next hitter will come to the plate. So you have to reset the D figure (in this case—the C figure if the other team is at bat) to the hitter *before* this spot in the lineup. Then, when you RUN 20, the figure will be incremented by 1, and Aaron will be at the plate. Since Traynor is the leadoff man, you can accomplish this by typing either D = 0 or D = 9. Now type RUN 20—but, before you ENTER it, clear the screen (with the CLEAR key, of course). Now hit the ENTER key.

(As an alternative, you could ENTER D = 1, and RUN 25, but you will lose the top part of the scoreboard until the next hitter comes up.)

Of course, you can also change pitchers by changing either line 55 or line 60, but it really doesn't make much sense unless your relief pitcher is much better than your starting pitcher. Since the pitcher tiring factor is based on the inning, the relief pitcher will be just as tired as the starter.

Where does all this data come from? The data development program takes raw baseball statistics and converts them into data for use in Historic Baseball. As a matter of fact, it prints out the hitters' DATA lines, except for the line numbers, and it also shows you exactly how the pitchers' program lines should read.

The hitter's portion of the program is fairly straightforward. The player's official at-bats, hits, doubles, triples, home runs, walks, and strikeouts are input. The program subtracts strikeouts from at-bats, computes a batting average based on this figure, multiplies by 500 to get the H figure, and then computes the J, K, L, M, and N figures. Then it prints out the DATA line.

The procedure for pitchers is somewhat different, because the program calls for major-league figures on batting average, average walks per game, and average strikeouts per game. This information is all contained in a table on page 23 of the *Encyclopedia of Baseball.* For specific seasons, you can

consult almanacs and such specialized publications as the annual *Sporting News Dope Book*.

The program computes the batting average (again, not including strikeouts) against the specific pitcher and divides this by the major-league batting average to produce a factor. It does the same with his average numbers of walks and strikeouts per nine innings, and then it prints out the program line.

One word of caution: Memory is very tight if you condense this for 4K Level I. If you have Crossetti, Lazzeri, DiMaggio, Yastrzemski, and others on the same team, your TRS-80 may respond with a SORRY. Abbreviate names, as necessary, the way they often do in the newspaper box scores: Yastrzemski becomes "Yaz," and so forth.

Although you can't do anything but watch the game and use a pinch-hitter here and there, there is room for a lot of experimentation between games. In my All-Star series, for example, each team has a 23-player roster, including five pitchers, so I've been using different combinations of players and different batting orders for each game. In some games, I've used a designated hitter; in others, I've had the pitchers batting for themselves.

As in real baseball, pinch-hitters are used more often for pitchers than for anyone else—especially since, in this program, the other players are all outstanding hitters. And, unless you grossly violate the rules of baseball, you do have to bring in a reliever after you've pinch-hit for the pitcher.

I haven't done this yet, but it would be possible to simulate the effects of righty vs. righty, lefty vs. lefty, etc., by juggling with the data. Against a left-handed pitcher, for example, you could arbitrarily subtract 5 from each left-handed hitter's H figure (lowering the batting average by 10 points) and add 5 to each right-handed hitter's H figure. But, for consistency's sake, this would require changing all the DATA lines again if a right-handed relief pitcher comes in.

I don't know how meaningful Historic Baseball is. Like any sport, baseball involves many intangibles which cannot be computerized (at least, not accurately), such as the ability of some players to hit in the clutch, the effect of daring base-running in helping to stimulate or disconcert the offense, and so on.

For lack of memory, many of baseball's myriad possibilities have been left out of the program. A runner attempting to score from second on a single might be thrown out at home plate, while another runner goes from first to third on the same play and the batter takes second on the throw-in. Outs will be specific: grounders, outfield flies, line drives, or pop-ups, and the probabilities will be pegged to the hitter's statistics. A power hitter like Babe Ruth will be much more likely to hit a long fly ball than a singles hitter like Pie Traynor. There will be wild pitches, passed balls, errors—based on the ac-

tual fielding percentages of the players involved—and there will be more of the actual suspense of a real baseball game.

The printout might say, for example, IT'S A LONG FLY BALL TO LEFT FIELD, and then, after a time delay, IT'S A HOME RUN! or IT'S OFF THE WALL FOR A DOUBLE or IT'S CAUGHT AT THE WALL FOR OUT #3.

That, however, is still in the future. In the meantime, I'm enjoying the 4K version and I imagine that anyone who is interested in both computers and baseball also will enjoy it.

Program Listing. *This listing was originally in Level I.*

```
 5 O = 0:
   I = 1:
   R = 0:
   C = 0:
   D = 0:
   V = 0:
   X = 0:
   Y = 0
10 INPUT "WHO BATS FIRST? 1-A. L.    2-N. L. ";A:
   CLS
20 RESTORE :
   GOSUB 5000:
   IF V > 0
    THEN
     GOSUB 7000:
     V = 0
25 IF A = 1
    THEN
     40
28 D = D + 1:
   IF D = 10
    THEN
     D = 1
30 W = D + 9:
   GOTO 45
40 C = C + 1:
   IF C = 10
    THEN
     C = 1
42 W = C
45 FOR Z = 1 TO W:
    READ B$,H,J,K,L,M,N:
    NEXT Z
50 B = 0:
   S = 0:
   GOSUB 5014:
   IF A = 1
    THEN
     60
55 H = INT(.902 * H):
   J = INT(.747 * J):
   K = INT(1.62 * K):
   GOTO 65
60 H = INT(.958 * H):
   J = INT(.553 * J):
   K = INT(1.31 * K)
65 H = H + 2 * (I - 4):
   L = INT(L * H):
   M = INT(M * H):
   N = INT(N * H)
70 GOSUB 5065:
   PRINT @640,"HERE COMES THE PITCH":
   GOSUB 9000
75 G = 500 - H:
   Z = RND(500 + J + K):
   IF Z < = K
    THEN
     800
80 IF Z < = K + J
    THEN
     900
85 IF Z < = K + J + G
    THEN
     960
90 IF Z < = K + J + G + L
    THEN
```

Program continued

```
      700
 95 IF Z < = K + J + G + L + M
     THEN
      600
100 IF Z < = K + J + G + L + M + N
     THEN
      500
105 PRINT "IT'S A SINGLE":
    W = RND(10):
    ON R + 1 GOTO 920,925,290,300,295,305,310,320
290 IF W > 6
     THEN
      930
295 V = 1:
    GOTO 920
300 IF W > 6
     THEN
      935
305 V = 1:
    GOTO 925
310 IF W > 6
     THEN
      V = 1:
      GOTO 930
315 V = 2:
    GOTO 920
320 IF W > 6
     THEN
      V = 1:
      GOTO 20
325 V = 2:
    GOTO 925
500 PRINT "IT'S A HOME RUN!":
    V = 1
505 ON R + 1 GOTO 525,510,510,515,510,515,515,510
510 V = V + 1:
    GOTO 525
515 V = V + 2:
    GOTO 525
520 V = V + 3:
    GOTO 525
525 R = 0:
    GOTO 20
600 PRINT "IT'S A TRIPLE":
    IF R = 0
     THEN
      945
605 IF R = 7
     THEN
      V = 3:
      GOTO 945
610 IF (R = 3) + (R = 5) + (R = 6)
     THEN
      V = 2:
      GOTO 945
615 V = 1:
    GOTO 945
700 PRINT "IT'S A DOUBLE":
    W = RND(10):
    ON R + 1 GOTO 940,710,715,720,715,720,725,730
710 IF W > 6
     THEN
      950
715 V = 1:
    GOTO 940
720 IF W > 6
     THEN
      V = 1:
      GOTO 950
725 V = 2:
    GOTO 940
```

Program continued

```
730 IF W > 6
     THEN
      V = 2:
      GOTO 950
735 V = 3:
    GOTO 945
800 W = RND(100):
    IF W < 86
     THEN
      804
802 PRINT "IT'S A FOUL BALL":
    IF S = 2
     THEN
      PRINT "THE COUNT HOLDS":
      GOTO 70
804 S = S + 1:
    PRINT "IT'S STRIKE";S:
    IF S = 3
     THEN
      GOSUB 3220:
      GOTO 20
810 GOTO 70
900 B = B + 1:
    PRINT "IT'S BALL";B:
    IF B = 4
     THEN
      910
905 GOTO 70
910 PRINT "THAT'S A WALK":
    IF R = 7
     THEN
      V = 1
915 ON R + 1 GOTO 920,925,925,935,930,935,935,935
920 R = 1:
    GOTO 20
925 R = 3:
    GOTO 20
930 R = 5:
    GOTO 20
935 R = 7:
    GOTO 20
940 R = 2:
    GOTO 20
945 R = 4:
    GOTO 20
950 R = 6:
    GOTO 20
960 IF (O < 2) * (R / 2 < > INT(R / 2))
     THEN
      980
962 PRINT "IT'S AN OUT":
    GOSUB 3220:
    GOTO 20
980 W = RND(100):
    IF W < 86
     THEN
      962
985 PRINT "IT'S A DOUBLE PLAY":
    GOSUB 3220:
    GOSUB 3220
986 ON INT(R / 2 + 1) GOTO 990,945,988,988
988 IF O = 2
     THEN
      V = 1
989 IF R = 7
     THEN
      945
990 R = 0:
    GOTO 20
1505 DATA 2B-C.GEHRINGER,167,779,185,.202,.051,.065
```

Program continued

```
1510 DATA LF-TY COBB,193,650,244,.173,.07,.028
1515 DATA RF-BABE RUTH,203,1298,630,.176,.047,.249
1520 DATA 1B-LOU GEHRIG,188,1047,411,.197,.06,.181
1525 DATA SS-JOE CRONIN,166,809,401,.225,.052,.074
1530 DATA CF-JOE DIMAGGIO,171,685,240,.176,.059,.163
1535 DATA C-BILL DICKEY,163,641,205,.174,.037,.103
1540 DATA 3B-B.ROBINSON,153,455,448,.177,.03,.097
1545 DATA P-WALTER JOHNSON,132,305,513,.171,.075,.044
1600 DATA 3B-PIE TRAYNOR,165,388,171,.154,.068,.024
1605 DATA SS-HONUS WAGNER,174,558,282,.187,.074,.03
1610 DATA 2B-R.HORNSBY,195,744,365,.185,.057,.103
1615 DATA CF-WILLIE MAYS,176,730,553,.159,.046,.209
1620 DATA RF-HANK AARON,176,586,503,.173,.032,.183
1625 DATA LF-STAN MUSIAL,176,840,274,.2,.049,.131
1630 DATA 1B-BILL TERRY,183,509,319,.17,.051,.07
1635 DATA C-ROY CAMPANELLA,156,742,523,.153,.016,.208
1640 DATA P-C.MATHEWSON,123,425,605,.138,.033,.019
3220 O = O + 1:
     PRINT "THAT'S OUT #";O:
     IF O = 3
      THEN
       3235
3225 GOSUB 9000:
     RETURN
3235 A = A + 1:
     IF A = 3
      THEN
       A = 1
3240 O = 0:
     I = I + .5:
     R = 0:
     IF (I > 9)
      THEN
       3260
3245 GOSUB 5000:
     RETURN
3260 IF I = INT(I) * (X < > Y)
      THEN
       3280
3265 IF (A = 1) * (X > Y)
      THEN
       3280
3270 IF (A = 2) * (Y > X)
      THEN
       3280
3275 GOTO 3245
3280 PRINT "THE GAME IS OVER":
     END
5000 IF I = INT(I)
      THEN
       5003
5002 PRINT @15,"BOTTOM OF INNING"; INT(I):
     GOTO 5005
5003 PRINT @15,"TOP OF INNING";I
5005 IF A = 1
      THEN
       PRINT @35,"A.L. BATTING":
       RETURN
5008 PRINT @35,"N.L. BATTING":
     RETURN
5014 PRINT @64,"AMERICANS--";X:
     PRINT @108,"NATIONALS--";Y
5015 PRINT :
     PRINT B$;" AT BAT"
5020 ON R + 1 GOTO 5025,5030,5035,5040,5045,5050,5055,5060
5025 PRINT "NO RUNNERS":
     RETURN
5030 PRINT "RUNNER ON 1ST":
     RETURN
5035 PRINT "RUNNER ON 2ND":
```

```
     RETURN
5040 PRINT "RUNNERS ON 1ST & 2ND":
     RETURN
5045 PRINT "RUNNER ON 3RD":
     RETURN
5050 PRINT "RUNNERS ON 1ST & 3RD":
     RETURN
5055 PRINT "RUNNERS ON 2ND & 3RD":
     RETURN
5060 PRINT "BASES LOADED":
     RETURN
5065 GOSUB 9000:
     PRINT @640,"":
     PRINT @704,"":
     PRINT @768,"":
     PRINT @832,""
5070 PRINT @448,"BALL","STRIKE","OUT":
     PRINT B,S,O:
     GOSUB 9000:
     RETURN
7000 IF (I > 9) * (I < > INT(I))
      THEN
       7020
7005 PRINT @775,V;"RUNS SCORE"
7010 IF A = 1
      THEN
       X = X + V:
       RETURN
7015 Y = Y + V:
     RETURN
7020 IF (A = 1) * ((X + V) > Y)
      THEN
       V = Y - X + 1:
       GOTO 7035
7025 IF (A = 2) * ((Y + V) > X)
      THEN
       V = X - Y + 1:
       GOTO 7035
7030 GOTO 7005
7035 PRINT @775,V;"RUNS SCORE TO WIN THE GAME!":
     IF A = 1
      THEN
       X = X + 1
7040 IF A = 2
      THEN
       Y = Y + 1
7045 GOSUB 5014:
     END
9000 FOR Z = 1 TO 1500:
      NEXT Z:
     RETURN
```

GRAPHICS

Four Graphics Methods

TRSpirograph

Adventures in Roseland

Four Graphics Methods

by John Krutch

The TRS-80 is not distinguished for the high quality of its graphics. In fact, it has just about the lowest resolution graphics of any microcomputer on the market today. But there are so many ways to access the TRS-80's graphics capabilities, and it is so easy to take advantage of them, that it comes close to making up for the low resolution.

In this article, we will consider four methods of doing graphics on the TRS-80. We'll look at a single graphics routine which is handled in four different ways. Three of the variations will be written in Level II BASIC. The first involves SET; the second, POKE; and the third, PRINT. The last variation will be written in assembly language; it primarily invokes the block-move command LDIR. We will also make timing comparisons among the four methods, and we'll be arriving at conclusions which may be surprising to some readers.

First, let's examine graphics routines which use the BASIC statement SET.

SET Graphics

There are 128 × 48, or 6144, *pixels* (picture elements) on the TRS-80's video monitor. Any one of these pixels can be turned on (i.e., made white in contrast to the monitor screen's dark background) with the help of SET. Furthermore, any pixel that has been turned on can be turned off again with the RESET statement. Since a pixel can be anywhere on the screen along a two-dimensional grid, two coordinates, representing both dimensions, are always specified when you're turning a particular pixel on or off.

The format of the SET statement is

SET(X,Y)

where X is a number 0 to 127, representing any one of 128 *horizontal* positions along the monitor. Y is a number 0 to 47, representing any one of 48 *vertical* positions along the monitor. So X and Y together specify one pixel. For instance, SET(0, 0) turns on the pixel in the upper left-hand corner of the screen. SET(127, 0) turns on the pixel in the upper right-hand corner. SET(0, 47) turns on the pixel in the lower left-hand corner. SET(127, 47) turns on the pixel in the lower right-hand corner. RESET operates in a similar fashion.

SET and RESET are useful because they allow one or all of the 6144 pixels to be easily controlled through BASIC. If you need to make use of the TRS-80's high-resolution mode (such as it is), this is the way to go.

SET, however, has an annoying problem: It is very, very slow. Any graphics program which makes extensive use of it is bound to be slow as well.

If you want to see for yourself just how slow a SET graphics routine can be, type in Program Listing 1 and run it.

This program draws a chessboard on the display, line by tedious line. The routine is as straightforward as possible; it was not deliberately written to run slowly. If you time the routine, you should get the same figure I did: about 33 seconds from the first square of the chessboard to the sixty-fourth. In other words, BASIC is taking roughly half a second to draw each square. This is intolerably slow, especially since there are ways of doing the same routine without using SET which are orders of magnitude faster.

POKE to Video RAM

The tremendously useful POKE statement puts a byte of data directly into memory. POKE can therefore be utilized to do graphics when the data byte is placed in memory locations 3C00-3FFF (decimal 15360-16383). Locations 3C00-3FFF are the video RAM; any byte that finds its way here is picked up by hardware, translated into an ASCII or graphics character depending on the value of the byte, and placed into the corresponding location on the video display. The values that the graphics generation hardware interprets as graphics characters are hexadecimal 80-BF (decimal 128-191).

What we need to do in order to utilize POKE graphics, then, is decide which graphics characters we need for our graphics routine, and then POKE them into the appropriate locations in video RAM. Graphics character 191 is a solid rectangle; several of them will do nicely for the white square of a chessboard. Program Listing 2 POKEs character 191 into the video RAM to produce a chessboard on the screen. This POKE chessboard, incidentally, is the same as the SET chessboard in every respect; the only way you can tell the difference between the two is by examining the listings.

This one is much faster than the SET routine; my reading is about eight seconds, four times faster than the SET program. But we can do it faster still. Quite a bit faster, as a matter of fact. And we can do it in BASIC.

Printing Graphics Strings

STRING$ is a handy function which lets you create a lengthy string of characters with just a few keystrokes. For instance,

```
PRINT STRING$(20, "X")
```

gives you a string of twenty Xs. But the character to be incorporated into the string doesn't have to be specified directly, as in the example above; it can be specified by using its ASCII (or character) code. This feature makes it possible to create strings of graphics characters. For example,

```
A$ = STRING$(100,136)
```

puts a string of 100 of the graphics character 136 into variable A$. A$ can now be printed and the string of graphics characters will appear on the

display. A$ can also be concatenated just like any other string.

We make use of the STRING$ function combined with PRINT in the program in Program Listing 3. After defining A, B, C, and D as string variables, a string of eight of the graphics character 191 is put into B and eight blanks are put into C. (Graphics character 191, you'll remember, is the solid rectangle we used in the POKE program.) Eight of the character 191, when printed side by side, form the top half of a white chessboard square Similarly, eight blanks would form the top half of a black square.

A horizontal row of a chessboard can either begin with a white square and end with a black one, or begin with a black square and end with a white. Line 60 concatenates B and C eight times to form the top half of a white-to-black row and stores this new string of characters in A. Line 70 does the same thing in a different order to form the top half of a black-to-white row; this is stored in D. Line 80 prints A and D the number of times needed to create the chessboard on the screen. Notice that the last few positions of the chessboard are POKEd in instead of printed directly; if the last position (PRINT@ 1023) were printed, the display would scroll, and the chessboard would be ruined. I know of no way to get around this in BASIC.

The amazing thing about this little program is the rapidity with which it executes. It will print a chessboard that fills the screen, just like the boards that were made with SET and POKE, but it does so in one second flat. This is some 33 times faster than the SET program! There's no getting around it; if you're writing a BASIC program which involves graphics routines, you should use PRINT STRING$ statements if you're interested in getting classy, high-speed graphics.

PRINT statements are the fastest way a chessboard (or almost any graphics routine) can be done in BASIC. But, of course, there is one further way to put a chessboard on the screen, and it is *much* faster than anything we've looked at so far.

Machine Language

Machine-language programs are fast because they take direct control of the microprocessor. There is no need for a BASIC interpreter to serve as middleman.

As you might expect, a chessboard-drawing routine written in assembly language (which will, of course, be assembled into machine language) will be lightning-fast compared to even the fastest of our BASIC programs.

Program Listing 4 is a listing of such a routine. The program takes the familiar graphics character 191 (hexadecimal BF) and puts it into the first eight locations in the video RAM. Then it puts character 128 (hexadecimal 80), which is a blank, into the next eight video RAM locations.

At this point, what you would see on the screen is the top half of the first

two squares of the chessboard (i.e., the top half of a white square and the top half of a black square). The program so far has operated very much like the BASIC POKE program in Program Listing 2. But now a new phase begins. The program takes those first sixteen bytes in the video RAM and uses them as a "bootstrap" segment. With the block-move instruction LDIR, the bootstrap bytes are put into the next 16 memory locations, then the next 32, and 64, and so on, until the entire video RAM is filled with chessboard graphics characters.

The BASIC program in Program Listing 5 incorporates the assembly-language program of Program Listing 4 as a USR routine. This program will not operate under Disk BASIC, because of differences in the way Disk BASIC and Level II BASIC handle USR.

The assembly-language program (which is completely relocatable) can be made into an object file with the help of Radio Shack's Editor/Assembler; you can then load it into the computer with the SYSTEM command, or the object code can be put directly into memory with T-BUG, or you can just type in the BASIC program in Program Listing 5.

If you load either of these programs into your computer, you'll find (surprise, surprise!) that machine-language graphics are *fast*. The chessboard will appear on the display in the blink of an eye.

But "blink of the eye," come to think of it, is a rather imprecise term. Just how much faster, exactly, is the machine-language routine than the various BASIC routines? Let's figure it out.

Each of the instructions in the Z-80's instruction set, while executing, lasts a certain number of *T-states* (clock cycles). Zilog, the manufacturer of the Z-80, has provided tables which show the number of T-states that each instruction takes up.

For instance, referring to Program Listing 4, the instruction

```
LD HL,3C00H
```

in line 120 of the listing requires 10 T-states to execute. The instruction in the next line, line 130,

```
LD (HL),0BFH
```

also requires 10 T-states to execute, but since it's part of a loop which cycles eight times, it takes up a total of 80 T-states.

If we go through the entire listing in this way, calculating the number of T-states needed for each instruction, we find that the program requires a total of 21,820 T-states to get to line 550, where the HALT loop is located.

Now, since the TRS-80 runs at a clock speed of 1.774 MHz, or 1,774,000 cycles per second, one T-state is equal to about 0.6 microseconds (0.0000006 seconds). 21820 T-states times 0.6 microsec. = 0.013 sec. In other words, it takes the machine-language routine only about one-hundredth of a second

to draw the chessboard on the screen!

The Results

The timing comparisons we've made are summarized in Table 1. As you can see, the machine-language routine is the hands-down winner, being 100 times faster than its nearest competitor, and fully 3300 times faster than the SET routine.

GRAPHICS METHOD	TIME REQUIRED
SET	33 sec.
POKE	8 sec.
PRINT	1 sec.
Machine-language	0.01 sec.

Table 1. *Time taken by four graphics methods to draw a chessboard on the display.*

The most interesting thing about our timing comparisons is the fine performance of PRINT graphics. Although PRINT isn't in the same league as machine-language graphics as far as speed is concerned, it's a whole lot closer than SET and POKE. And PRINT statements have a distinct advantage over machine language when it comes to actual programming, since graphics routines are easier to write in BASIC using PRINT statements than they are in assembly language.

Program Listing 1

```
  1 :
    '   SET GRAPHICS DEMO
 10 CLS
 20 FOR I = 1 TO 4
 30  FOR J = 1 TO 6
 40   FOR K = 1 TO 4
 50    FOR L = N TO N + 15
 60     SET(L, Y)
 70     NEXT L
 80    N = N + 32
 90    NEXT K
100   N = 0:
      Y = Y + 1
110   NEXT J
120  FOR M = 1 TO 6
130   FOR P = 1 TO 4
140    N = N + 16
150    FOR Q = N TO N + 15
160     SET(Q, Y)
170     NEXT Q
180    N = N + 16
190    NEXT P
200   N = 0:
      Y = Y + 1
210   NEXT M
220  NEXT I
230 GOTO 230
```

Program Listing 2

```
  1 :
    '   POKE GRAPHICS DEMO
 10 CLS
 20 L = 15360
 30 FOR I = 1 TO 4
 40  FOR J = 1 TO 2
 50   FOR K = 1 TO 4
 60    FOR M = 1 TO 8
 70     POKE L, 191
 80     L = L + 1
 90     NEXT M
100    L = L + 8
110    NEXT K
120   NEXT J
130  FOR N = 1 TO 4
140   FOR P = 1 TO 2
150    L = L + 8
160    FOR Q = 1 TO 8
170     POKE L, 191
180     L = L + 1
190     NEXT Q
200    NEXT P
210   NEXT N
220  NEXT I
230 GOTO 230
```

Program Listing 3

```
  1 :
    '   PRINT GRAPHICS DEMO
 10 CLEAR 200
 20 CLS
 30 DEFSTR A - D
 40 B = STRING$(8, 191)
 50 C = STRING$(8, " ")
 60 A = B + C + B + C + B + C + B + C
```

```
 70 D = C + B + C + B + C + B + C + B
 80 PRINT A;A;D;D;A;A;D;D;A;A;D;D;A;A;D;C;B;C;B;C;B;C;
 90 FOR I = 16376 TO 16383
100  POKE I, 191
110  NEXT I
120 GOTO 120
```

Program Listing 4

```
                00100 ;MACHINE-LANGUAGE GRAPHICS DEMO
4A00            00110          ORG     4A00H
4A00 21003C     00120          LD      HL,3C00H
4A03 36BF       00130 LOOP1    LD      (HL),0BFH
4A05 23         00140          INC     HL
4A06 7D         00150          LD      A,L
4A07 FE08       00160          CP      08H
4A09 20F8       00170          JR      NZ,LOOP1
4A0B 3680       00180 LOOP2    LD      (HL),80H
4A0D 23         00190          INC     HL
4A0E 7D         00200          LD      A,L
4A0F FE10       00210          CP      10H
4A11 20F8       00220          JR      NZ,LOOP2
4A13 21003C     00230          LD      HL,3C00H
4A16 11103C     00240          LD      DE,3C10H
4A19 011000     00250          LD      BC,16
4A1C EDB0       00260          LDIR
4A1E 21003C     00270          LD      HL,3C00H
4A21 11203C     00280          LD      DE,3C20H
4A24 012000     00290          LD      BC,32
4A27 EDB0       00300          LDIR
4A29 21003C     00310          LD      HL,3C00H
4A2C 11403C     00320          LD      DE,3C40H
4A2F 014000     00330          LD      BC,64
4A32 EDB0       00340          LDIR
4A34 21083C     00350          LD      HL,3C08H
4A37 11803C     00360          LD      DE,3C80H
4A3A 012000     00370          LD      BC,32
4A3D EDB0       00380          LDIR
4A3F 21803C     00390          LD      HL,3C80H
4A42 11A03C     00400          LD      DE,3CA0H
4A45 012000     00410          LD      BC,32
4A48 EDB0       00420          LDIR
4A4A 21803C     00430          LD      HL,3C80H
4A4D 11C03C     00440          LD      DE,3CC0H
4A50 014000     00450          LD      BC,64
4A53 EDB0       00460          LDIR
4A55 21003C     00470          LD      HL,3C00H
4A58 11003D     00480          LD      DE,3D00H
4A5B 010001     00490          LD      BC,256
4A5E EDB0       00500          LDIR
4A60 21003C     00510          LD      HL,3C00H
4A63 11003E     00520          LD      DE,3E00H
4A66 010002     00530          LD      BC,512
4A69 EDB0       00540          LDIR
4A6B 18FE       00550 HALT     JR      HALT
4A00            00560          END     4A00H
00000 TOTAL ERRORS

HALT   4A6B 00550   00550
LOOP1  4A03 00130   00170
LOOP2  4A0B 00180   00220
```

Program Listing 5

```
 1 :
 '              BASIC + MACHINE LANGUAGE GRAPHICS DEMO
10 FOR I = 18944 TO 19049
20  READ B
30  POKE I, B
```

Program continued

```
 40  NEXT I
 50 POKE 16526, 0:
    POKE 16527, 74
 60 PRINT "FOR A LOOK AT FAST MACHINE-LANGUAGE GRAPHICS,"
 70 PRINT "PRESS <ENTER>."
 80 A$ = INKEY$:
    IF A$ < > CHR$(13)
     THEN
      80
 90 X = USR(1)
100 DATA 33,0,60,54,191,35,125,254,8,32,248,54,128,35,125,254, 16,32
    ,248,33,0,60,17,16,60,1,16,0,237,176,33,0,60,17,32,60
110 DATA 1,32,0,237,176,33,0,60,1,64,0,237,176,33,8,60,17,128, 60,1,
    32,0,237,176,33,128,60,17,160,60,1,32,0,237,176,33,128
120 DATA 60,17,192,60,1,64,0,237,176,33,0,60,17,0,61,1,0,1, 237,176,
    33,0,60,17,0,62,1,0,2,237,176,24,254
```

TRSpirograph

by Ronald A. Balewski

The TRS-80 is a versatile instrument. It can be both amusing and enlightening. This program is a bit of both; it will give you a little better understanding of two graphic patterns.

To put the purpose of this program in a layman's terms, it draws spirograph patterns on the TRS-80's video screen. The mathematician, on the other hand, will tell you that it plots hypocycloids and epicycloids!

First, let us start with the *hypocycloid.* Hypocycloid is the name given to the curve traced out by some point on a circle which is rolling around the inside of a larger circle. Figure 1(a) should clarify this definition. The metric equations which describe this curve are as follows:

$$X = (A + B)\cos\Theta - B\cos(((A + B)/B)\Theta)$$
$$Y = (A + B)\sin\Theta - B\sin(((A + B)/B)\Theta)$$

Figure 1(b) is a typical pattern. As you can see, there are four points where the curve touches the circle. This would seem to indicate that the circumference of the track circle must be four times that of the rotating circle. (You may be interested to know that the rotating circle only revolved around its axis three times. I suppose the fourth expected rotation comes from the fact that the entire circle as a whole took a circular path. You can see this by imagining what happens as Figure 1(a) is drawn.) Using the formula for circumference, $C = 2\pi R$, we can see that the radius A of the track circle must be four times that of the rotating circle. To see this illustrated, use the values, say 20 and 5 for the track and rotating circle radii, respectively, and run this program. You will get a pattern similar to Figure 1 (b). *Epicycloid* is the name given to the curve traced out by some point on a circle which is rolling around the outside of another circle. (See Figure 2(a). The parametric equations which describe this curve are as follows:

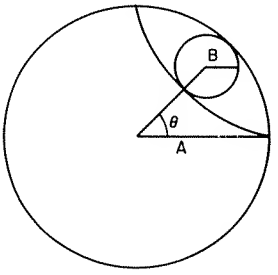


Figure 1(a)

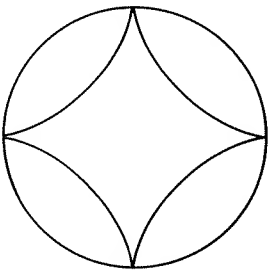


Figure 1(b)

$$X = (A + B)COSV - B\ COS(((A + B)/B)V)$$
$$Y = (A + B)SINV - B\ SIN(((A + B)/B)V)$$

This curve is on the outside as opposed to the inside of the track circle. Try using 20 and 5 again for the track and rotating circle radii, only this time request an epicycloid. You will see something similar to the drawing in Figure 2(b).

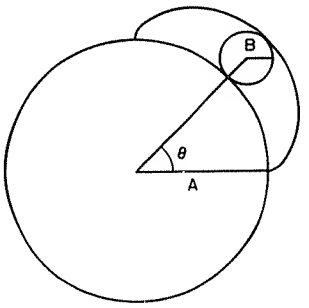


Figure 2(a)

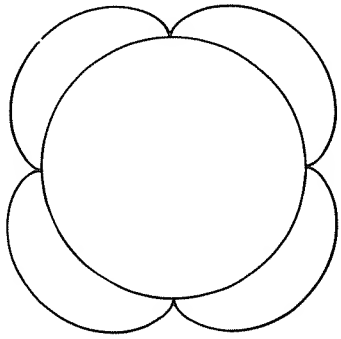


Figure 2(b)

One feature is still missing from our spirograph formula. With the above equations, we can only get variations of Figures 1(b) and 2(b). By moving the drawing point from the circumference of the rotating circle, we can get looping, leaf-type patterns. This would involve adding a constant to certain B values, while leaving the others alone. Elaborating on this, wherever B is used to dictate the motion of the revolving circle, it must be unchanged or this would throw off the desired number of nodes by effectively changing the radius of the rotating circle. On the other hand, Bs which are used to decide where the next point should be drawn can be changed to give you a different design with the same overall symmetry.

After some guesswork, I came up with four new parametric equations:

HYPOCYCLOID:

$$X = (A - (B + K))COS\Theta + (B + K)COS(((A - B)/B)\Theta)$$
$$Y = (A - (B + K))SIN\Theta - (B + K)SIN(((A - B)/B)\Theta)$$

EPICYCLOID:

$$X = (A + (B + K))COS\Theta - (B + K)COS(((A + B)/B)\Theta)$$
$$Y = (A + (B + K))SIN\Theta - (B + K)SIN(((A + B)/B)\Theta)$$

Once I had these monstrous equations, it was a simple matter to write the program to go with them. All the program does is step from 0 to $2\pi N$, where N is the number of times to go around the track circle. For each step, X and Y are calculated, scaled, and plotted.

A few words on operating the program: After the identification message, you will first be asked for the radii of the track and rotating circles. You can

use any values here, as the graph will be automatically scaled to fit the screen.

Next you will be asked for a pen displacement (our old friend constant k). Zero will draw a perfect hypocycloid or epicycloid. A positive value will draw a design with looping edges. The bigger the value, the bigger the loop. A negative value will draw a smoother curve. If you enter the negative of what you had entered for the radius of the rotating circle, you will get a circle (it should not be too difficult to see why). Try it! Next, you enter the number of revolutions around the track circle. Some designs take more than others. If you don't know how many you will need, enter a large number. You will be able to stop it when it's finished. Next, for the step size, about 0.1 should be good enough. A smaller number will give you a more filled-in line. K INCREMENT VALUE lets you gradually increment or decrement the constant k as the graph progresses. It makes for some unusual effects. Finally, if you want a hypocycloid, enter an H; for an epicycloid, enter an E.

When the drawing is complete, a period will appear in the top left corner of the screen. To draw the same pattern again, press the X. To draw a different pattern, press the space bar. If you decide that the drawing is complete before the entered number of revolutions, press the clear key. The drawing will stop and the period will appear. You can then either draw it again or make another drawing as stated above.

If you would like to see the X and Y values before they are scaled, insert the following line:

```
52 PRINT@0,X;Y;
```

To see the numbers after they are scaled, use this line:

```
56 PRINT@0,X;Y;
```

This program is fascinating!

track circle radius	rotating circle radius	pen displacement	revolutions	step size	k increment	design type
20	3	2	3	.025	0	H
30	3	7	1	.025	0	H
16	3	-1	3	.025	0	H
10	3	-1	3	.025	0	E
10	5	0	4	.025	.01	H
6	1	3	1	.025	0	E

Table 1. *Values for spirograph designs*

Program Listing

```
 1 REM  ******** TRSPIROGRAPH ********
 2 REM  BY RON BALEWSKI
 5 CLS :
   PRINT CHR$(23):
   PRINT @466,"TRSPIROGAPH":
   FOR K = 1 TO 900:
    NEXT
 6 CLS :
   INPUT "RADIUS OF TRACK CIRCLE";A:
   INPUT "RADIUS OF ROTATING CIRCLE";B:
   INPUT "RADIAL DISPLACEMENT OF PEN FROM ROTATING CIRCLE";K:
   INPUT "NUMBER OF REVOLUTIONS AROUND TRACK CIRCLE";NR:
   INPUT "SIZE OF EACH STEP AROUND TRACK CIRCLE (IN RADIANS)";ST
 7 INPUT "K INCREMENT VALUE";KI:
   PRINT :
   INPUT "WOULD YOU LIKE THE PATTERN BASED ON A HYPOCYCLOID (SMALL
   CIRCLE ROTATING INSIDE A LARGER ONE) OR AN EPICYCLOID (SMALL CIR
   CLE    ROTATING AROUND THE OUTER PERIMETER IF A LARGER ONE)";TY$
 8 TP = K
10 CLS
12 K = TP
15 Z = 3.14159 * 2 * NR
18 SX = (A + 2 * B + K) / 62:
   SY = (A + 2 * B + K) / 48:
   IF TY$ = "H"
    THEN
     SX = SX - (B / 62):
     SY = SY - (B / 48)
20 FOR TH = 0 TO Z STEP ST
21  K = K + KI
22  IF PEEK(14400) AND 2
     THEN
      80
25  IF TY$ = "E"
     THEN
      X = ((A + (B + K)) * COS(TH)) - ((B + K) * ( COS(((A
      + B) / B) * TH)))
30  IF TY$ = "E"
     THEN
      Y = ((A + (B + K)) * SIN(TH)) - ((B + K) * ( SIN(((A
      + B) / B) * TH)))
40  IF TY$ = "H"
     THEN
      X = ((A - (B + K)) * COS(TH)) + ((B + K) * ( COS(((A
      - B) / B) * TH)))
50  IF TY$ = "H"
     THEN
      Y = ((A - (B + K)) * SIN(TH)) - ((B + K) * ( SIN(((A
      - B) / B) * TH)))
55  X = (X / SX) + 62:
    Y = (Y / SY) / 2 + 24
60  IF (X > = 0 AND X < = 123) AND (Y > = 0 AND Y < = 47)
     THEN
      SET(X,Y)
70  NEXT TH
80 PRINT @0,".";
90 A$ = "":
   A$ = INKEY$:
   IF A$ = ""
    THEN
     90
100 IF A$ = " "
     THEN
      6:
     ELSE
      IF A$ = "X"
       THEN
        10:
       ELSE
        90
```

Adventures in Roseland

by Allan S. Joffe W3KBM

This general equation, J = a SIN X, if properly translated into a program that your TRS-80 can digest, paints a three-leafed rose onto your monitor screen. The Program Listing gives a programming possibility. After you have run the program and examined the scenery, the question "Why bother?" may come up.

Pattern After Pattern

For a partial answer, make the following changes and additions to the Program Listing.

```
 5 G=0
15 G=G+1:PRINT@50,G
30 R=35*SIN(G*J)
80 INPUT Z$
90 GOTO 10
```

You now have a program that produces pattern after pattern, because of the changing value of G, each time the program runs. Line 80 is merely a way to put in a controlled pause. When one pattern has been generated, you may examine it for as long as you wish, hitting ENTER to get the next one. The PRINT statement in line 15 is an index that will help you make a record of any pattern that happens to strike your fancy.

Running the revised listing, you will see that when G is an even number, the rose has petals equal to 2*G, and when G is odd, the petal count equals G. Note also that when G is odd, the figure is first traced and then retraced by the program. If you are going to run any number of these patterns, I suggest you alter the STEP in line 20 to read .035. This cuts the print time in half without too much damage to the image.

After you have played with the program for a bit, jump past the rose petal section by changing line 5 to read G = 29. Remember that as the patterns form, you can stop them as desired using SHIFT @. You will notice that some of the patterns are predominantly circular, while others are spirals. Some are cluttered looking and others are quite sharply defined.

You can expand them by setting the value in line 5 to such constants as 99, 199, or 299 to find new patterns. For more visual fun with your TRS-80, set the value of G in line 5 to 29. Line 20 should read:

```
20 FOR J=0 TO 3.14 STEP .035
```

This line eliminates some of the clutter you may have noticed in the patterns and also speeds up the printing of the image.

When G = 35 you see an image of five tangent circles. If G = 44 you have a gaggle of four circles. When G = 36 you see a stylized eagle inside a spiral segment. We already have index G as a guide. Add another index so you can see what I see in the following examples.

Change line 30 to read:

```
30 R = 35*SIN(G*J):PRINT@0,J
```

Start the program running again by setting G in line 5 to equal 28. The first time you run the program, G will equal 29. If you stop the pattern when J = 1.12, you should see what might be interpreted as a barbell weight.

If G = 33 and J = 1.575, you may see a dinosaur.

If G = 63 and J = 1.47, you will hopefully see Snoopy the dog (see Figure 1).

If G = 116 and J = 1.435, you will see a running dog.

Here are some other fantasies available by altering G: G = 143 is a stylized Darth Vadar, and G = 144 gives you a close approximation of the human eye as shown in a cross section of an anatomy book (see Figure 2).

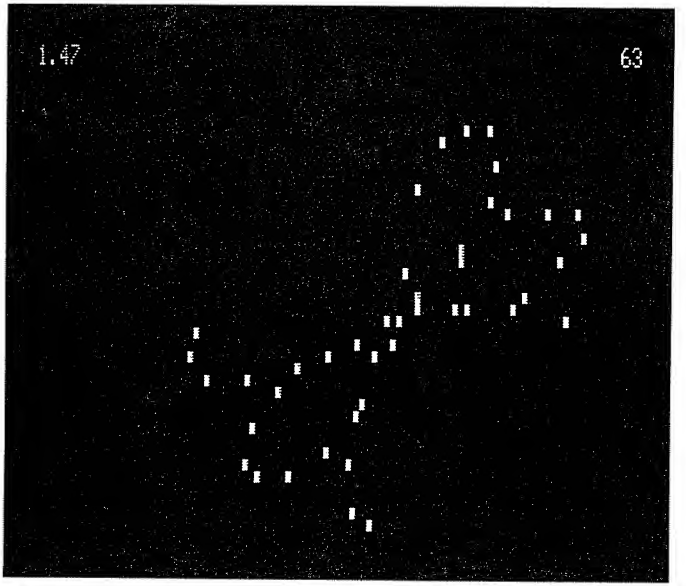


Figure 1. *Snoopy*

Negative Values

You can also use negative values for G. In this last image, let G = – 144. The pattern is identical except it has been rotated so that it is now the mirror image of the positive G input.

Since we are dealing with circular functions, this displacement can be left

to right as in this example, or top to bottom (G = 1 and G = −1).

You can also get a combination of shifts, such as both right to left and top to bottom, as when G = 36 or G = −36. There are times when altering the symmetry makes the image more realistic. For example, if G = 33 and J = 1.115, you see what looks like a running horse. If you alter line 30 to read:

```
30 R = 45*SIN(G*J)
```

the running horse becomes more realistic.

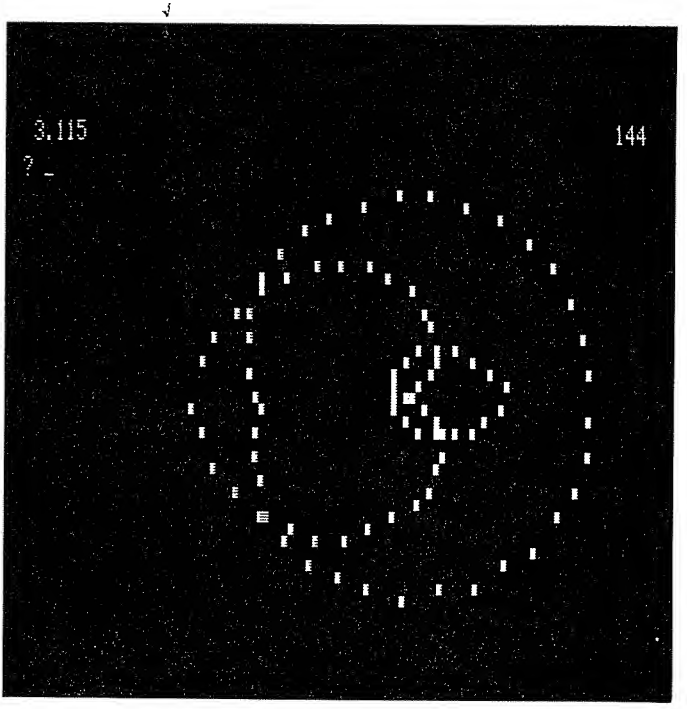


Figure 2. *The human eye*

Program Listing

```
10 CLS
20 FOR J = 0 to 6.28 STEP .0175
30 R = 35*SIN(3*J)
40 X = (R*COS(J)) + 64
50 Y = (R*SIN(J)) + 47
60 SET(X,47 - (Y/2))
70 NEXT J
```

HARDWARE

Punch Out Your Disks

Build a Snooper/ Snubber

Punch Out Your Disks

by Richard Taylor

For the price of a standard paper punch you can double your present disk storage. Just follow the simple steps detailed in this article and you can be reading and writing on both sides of your disks. I have punched out over 100 disks and only two of them have had defective second sides.

To get started you will need a pencil, a paper punch that catches its own punches, a tracing of a disk, and a smooth piece of paper. The tracing of the disk (which we shall call the "templet") can be made by photocopying a blank disk (*do not* photocopy a disk with data on it), cutting apart an unusable disk, or by making a tracing.

Try to use a stiff piece of paper or glue the copy to a piece of cardboard. Cut out the center hole, the oblong area below it, the write protect notch on the upper right edge, and the small hole near the center hole. The smooth piece of paper can be the backing from a peel-away label or something similar. The templet shown in the illustrations has two holes punched. This is just a convenience and is not needed to do the job.

The Second Hole

The only thing that prevents a Radio Shack disk drive from writing to the second side of your disks is that it needs a second small hole near the center so that it can find the sectors correctly. If you rotate your disk in its sleeve and watch the small hole, you will see an even smaller hole right in the disk. Soft-sectored disks have only one of these and the disk drive uses a light to "see" when this tiny hole passes by.

Our job is to punch a second hole in the sleeve so that when the disk is flipped over it will have a hole that allows the drive to see the tiny hole in the disk. The placement of the second hole does not have to be perfect. As long as the tiny hole can be seen through the new hole in the sleeve, everything will run correctly.

STEP 1: With the label of the disk in the upper left-hand corner, place the templet on the disk so that the small hole is positioned by the lower left side of the center hole. Line up all reference points. Using a pencil, trace the new small hole on the disk. Also trace the notch on the upper left edge (Figure 1).

STEP 2: Take the strip of smooth paper and insert it between the sleeve and the disk.

STEP 3: Using your thumb, make room for the punch by lifting the sleeve near the center hole (Figure 2).

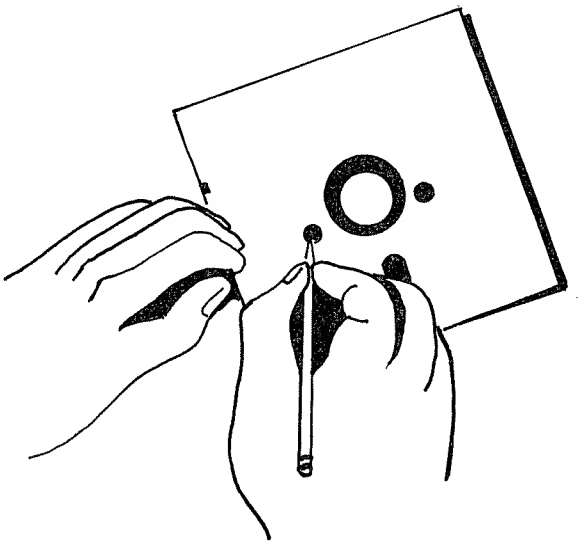


Figure 1

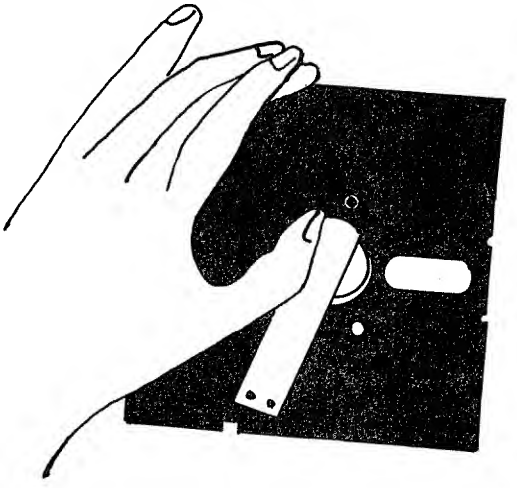


Figure 2

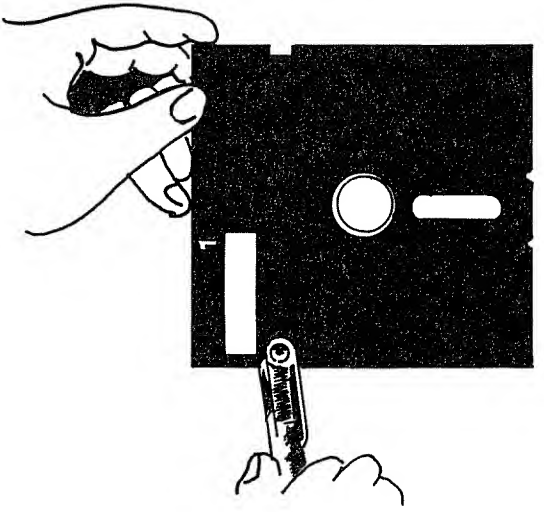


Figure 3

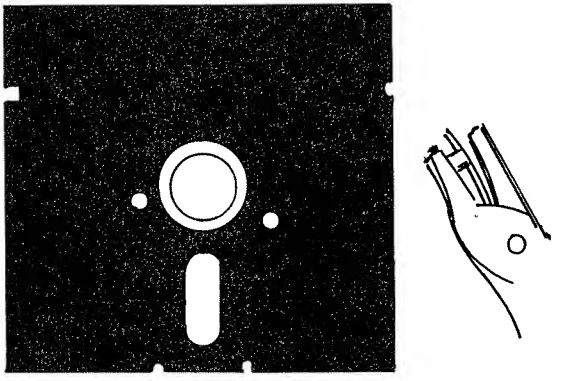


Figure 4

STEP 4: Insert the punch and line it up with the traced hole. Punch the hole.

STEP 5: Insert your finger where the punch was and check to see if the liner has been completely removed. In most cases it will not be. With your finger, push it up through the hole and tear it off.

STEP 6: Repeat steps 1 through 5 on the second side of the disk.

STEP 7: Punch the new notch near the bottom of the label (Figure 3).

That's all there is to it. Figure 4 shows you what your new disk should look like. Labels can be placed in the upper left corner with no problems. Any problems with the new side will show up immediately, just as they would with a new disk. There is no need to treat this new style any differently. All of my disks are double-sided and, while I was unsure at first, I now have no fear of using the second side for the most important programs and data. In the early days there were problems involving bulk erasing. Now we have TRSDOS 2.2, 2.3, and NEWDOS. All of these operating systems will back up over a disk that contains data without requiring bulk erasing.

Build a Snooper/Snubber

by Philip O. Martel KA1GK

The relay that controls the TRS-80's cassette player is subject to high voltage across its contacts when it's turned off.

Radio Shack gives the relay some protection with two 75-volt zener diodes across the contacts, but this isn't always enough. The relay sometimes welds shut.

When this happens, you can remove the remote plug from the recorder and operate it manually. Eventually the relay unsticks. Or you can try the snooper/snubber.

Easy to Build

The snooper/snubber, a small easy-to-build electronic circuit, monitors the TRS-80's cassette interface and gives the relay extra protection. The snubber circuit (Figure 1) gives the motor current some place to go when the relay contacts open, so that the current doesn't try to jump across the contacts and weld them together.

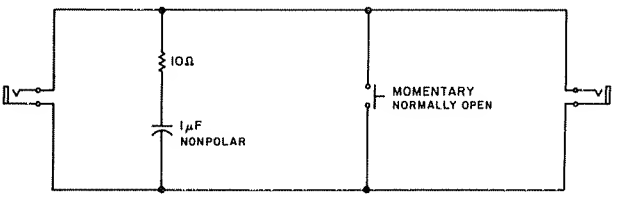


Figure 1. *The Snubber Circuit*

The snubber can be placed anywhere between the relay contacts and the cassette recorder motor. To avoid breaking any of Radio Shack's seals, I put the snubber circuit in a small box, plug the remote from the TRS-80 into the box, and run a jumper cable with subminiature phone plugs from the box to the recorder. In addition, a push-button switch across the circuit lets me advance the tape in play mode.

The snooper (Figure 2) is a simple means of monitoring the audio signals into and out of the TRS-80. The snooper consists of five miniature phone jacks, a crystal earphone, a double-pole double-throw (DPDT) center-off switch, and some shielded cable.

Two of the jacks accept the earphone and auxiliary plugs from the TRS-80. Two others pass out the same signals to the cassette recorder via jumper cables with miniature plugs on each end. The fifth jack passes one of

the two signals to an external device such as an amplifier. You'll need one, if you are running programs that produce sound.

The DPDT switch determines which of the two signals—the earphone signal (to the TRS-80) or the auxiliary signal (from the TRS-80)—is passed to the fifth jack and to the crystal earphone. The crystal earphone outputs a low-volume signal, audible, but not loud enough to require a volume control.

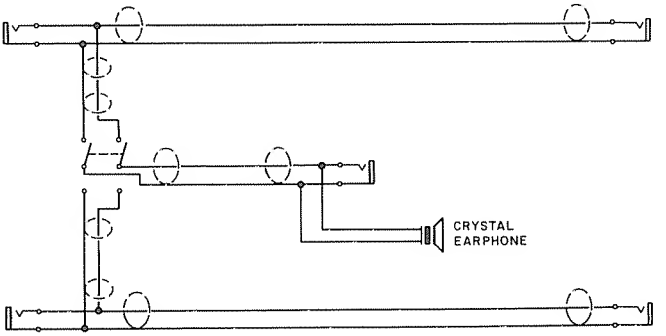


Figure 2. *The Snooper Circuit*

Construction

If you have some experience in electronics, you can build the snooper/snubber from the schematic diagrams. If not, I've provided some guidelines.

Build your snooper/snubber in a plastic box. If it's metal, you may encounter ground loops. These cause a loud, low-pitched buzz on recorded tapes. A box about 2 × 3 × 4 inches is a good size. You can use one-half this size, but, unless you like repairing watches or constructing ships in bottles, it's likely to prove frustrating.

You can lay out the components any way you like. Mine has the jacks for the TRS-80's cable in front, the jacks for jumper cables in back, the fifth jack on one side, and the two switches and earphone on top.

Drill the holes and mount the components loosely. (Miniature jacks take 1/4-inch holes and subminiature jacks take 3/16-inch holes.) How you mount the earphone depends on its shape. If the earphone is flat, drill several small holes and glue the earphone behind them. If the earphone has a roughly cylindrical earplug, drill a hole to fit the earplug and glue the earplug to the box.

Take the shielded wire and run it between the jacks and switches to measure how much is needed. Leave an extra inch and a half on one end of each piece of wire as it is cut. This may seem like a lot, but it is much better

to stuff any extra length of wire into the box than to come up a tenth of an inch short.

Before you solder the circuit together, remove all the components from the box and put it well away from your soldering iron. The affinity that plastic boxes have for hot soldering irons, you cannot believe.

Soldering

The wiring is straightforward, but most of the parts, especially the jacks, are fairly small. You aren't going to get all the wires from the shield through those little holes in the jacks. Cut off about half of the wires very close to the insulation and things will go much easier.

Make sure that all the remaining strands of the shield are twisted together. One tiny, almost invisible strand of wire can short out one of the signals. This condition is not likely to damage anything, but the time spent trying to track down a short can be frustrating.

The DPDT switch (Figure 3) has a 3×2 array of contacts on the bottom. All the shields should be connected to one set of three contacts, and all of the center wires to the other set of three contacts.

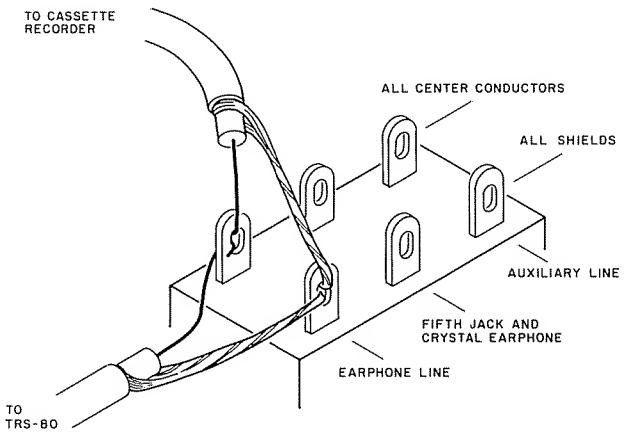


Figure 3. *The DPDT Switch*

The shielded wires can be run from one jack to the other and then to the switch, or from one jack to the switch and then to the other. I recommend the second approach, since it puts the point where the two shielded wires join at the switch, which usually has larger contacts than the jacks.

The specific values given for the resistor and capacitor are not critical. Anything within a factor of about ten should work fine. That is, the capacitor should be between about 0.3 microfarads (uF) and 3 uF and the resistor

should be between about 3 Ohms (Ω) and 30 Ω. The capacitor and resistor are soldered together by one lead and soldered to the normally open-push-button switch by the other (Figure 4).

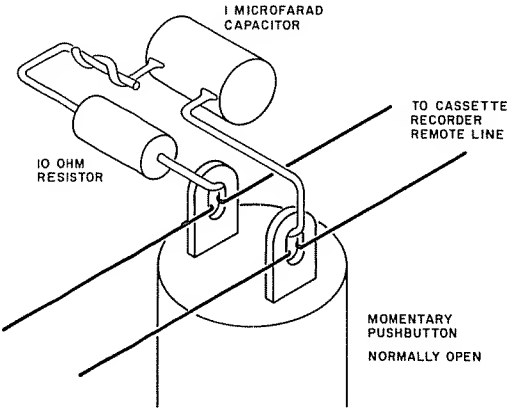


Figure 4. *The Capacitor and Resistor*

The earphone should be a high impedance type. A crystal earphone is specified, but any type with an impedance of 10 kΩ (10 kilohms = 10,000 Ohms) or more will work. The high impedance minimizes loading and results in a fairly low volume.

Once you have the snooper/snubber assembled, normal use of the cassette recorder will test it. If you would like to give the snubber a thorough test, run the program shown in Program Listing 1. It will turn the cassette motor on and off repeatedly with a period of about five seconds. I ran this program for more than 1000 cycles of the relay with no trouble. Not bad, considering that the relay had failed the first day I used my TRS-80.

Program Listing 1. *Test Program*

```
10 OUT 255,4
20 FOR I = 1 TO 1000: NEXT I
30 OUT 255,0
40 FOR I = 1 TO 1000: NEXT I
50 GOTO 10
```

HOME APPLICATIONS

Car Pool

Doctor Your Records

Computacar

Bio-Bars: Biorhythms in Bar Graph Form

Car Pool

by Walter K. McCahan

While driving to work and contemplating nontrivial uses for my TRS-80 microcomputer, I noticed that almost every car on the road was occupied by only one person. Keeping in mind the high cost of gasoline and other automobile operating expenses, I felt that a bit of research was in order.

After a hasty and short survey of a local office building, I found that the overwhelming reason that car pools had not been formed was the employees' lack of knowledge as to which other employees lived close to them and were available to form a pool. I also found that the employer was willing to pay a healthy sum to provide employees with the information with which to form car pools. To attack and solve this problem I devised the following plan of action:

1. Find the home address of each employee.
2. Reduce these addresses into zones.
3. Write a program to match employees that live close to one another.
4. Give each employee a list of these other nearby employees and instruct them to form car pools.

Step 1.

Gathering personal information from the employees in this building was not a difficult task and was accomplished by the letter shown in Figure 1. The letter was copied on the company's copy machine and was distributed with each employee's payroll check. I found that since there was a direct savings to them, the employees were quick to complete the information and return the questionnaire.

Step 2.

After looking into several methods of relating addresses to locations, including the name of the town and zip code, it was decided that a completely independent method of zoning must be developed.

Using a map of the local area (Harrisburg, PA), I produced a grid of 16 sections (see Figure 2). Each zone on this grid represents 225 square miles, so that the entire grid represents the 3600 square miles centered on Harrisburg, an area of about 1.5 million people. Each employee was provided with a copy of this map and was instructed to enter the proper zone on the questionnaire.

Fellow Employees:

In these times it is no longer economical for each employee to drive his or her car to work alone.

In an effort to conserve gasoline (and therefore money), your company is providing a computer service that will match your name with other employees that work with you so that you can form a car pool.

Attached you will find a map of our local area which has been divided into 16 zones. You will also find below a form that is to be clipped and returned to the personnel department after it is completed.

All items on the form are self-explanatory except for the one labeled "Zone." In this blank you should enter the number of the zone in which you live, according to the numbers on the map. If you do not live within one of the zones, enter the zone nearest to your home.

After all of the forms have been returned and entered into the computer you will be furnished with a list of names and instructions on how to set up your car pool.

Sincerely,
The Management

CAR POOL QUESTIONNAIRE

NAME _____
STREET ADDRESS _____
CITY & ZIP _____
TELEPHONE NO. _____
ZONE _____
SHIFT _____
BUILDING _____

Figure 1. *Gathering Employee Information*

Step 3.

To accomplish matching employees, the Level II TRS-80 program in the Program Listing was written. Before getting into a general discussion of the program, it would be well worth mentioning a somewhat unorthodox method of saving and loading data to and from tape. When first looking at the program, it is surprising that only one array is dimensioned—line 50. This is all that is necessary since all information for each record is concatenated into one string called N$. This is done at lines 390 or 410. By using this method, record and retrieve times to and from cassette tape are greatly reduced.

Step 4.

A match was printed for each person who participated in the project and a copy was sent to each employee along with the letter shown in Figure 3.

Conclusion

The results of this venture were very gratifying, in that the number of cars in the company's parking lot have decreased substantially while the number of employees working in the building has increased. The program has since been used by others, including one large company who ran the program through several times, entering only those who worked on one shift in each run. It has also been used by a company for the registrants of a seminar they were holding.

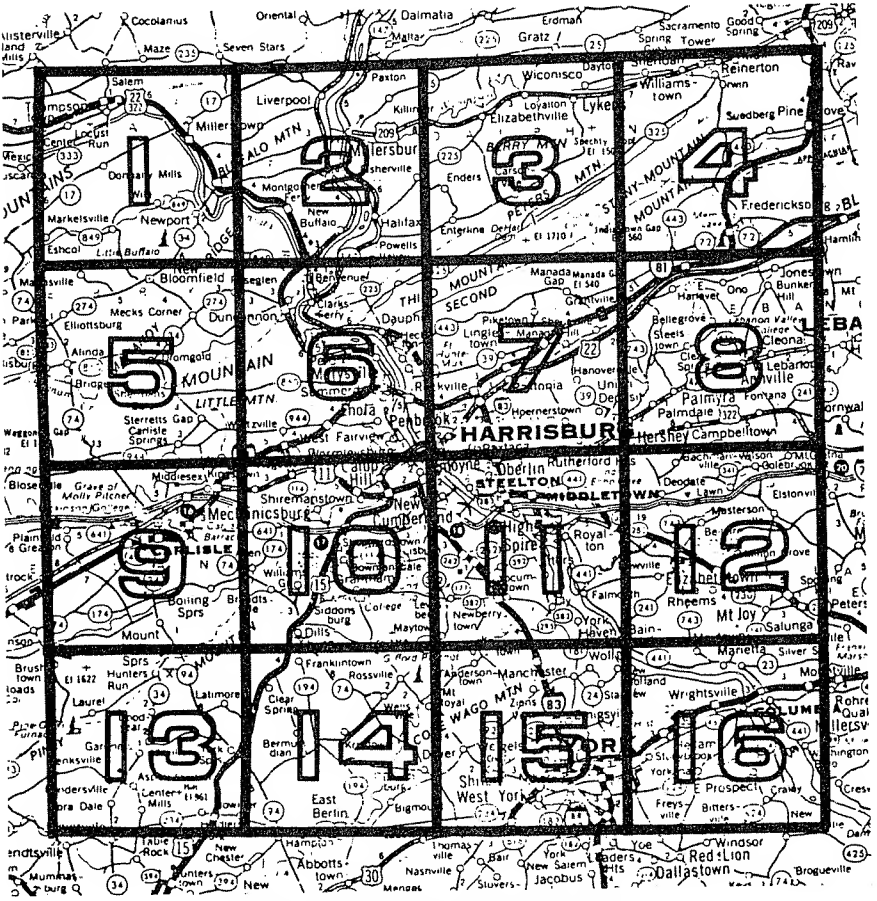


Figure 2. *Grid Map of Harrisburg, PA area*

Dear Fellow Employee:

About two weeks ago you were asked to complete and return a questionnaire which was concerned with car pooling.

Attached is a computer printout of the results of this project as they relate to you.

On the printout you will find a list of people who represent the optimum prospects for forming a car pool—these people are under the category of first choices—and all live quite close to you. The second category is those people who are second choices. These people do not live as close as those in the first category, but do live close enough to make a car pool worthwhile.

With this list we have provided you with the basic information needed to form your own car pool—the rest must be worked out by you. The easiest way to form a pool is to start calling your fellow employees starting from the top of the list (each employee's telephone number is given) and make your own meeting arrangements.

Sincerely,
The Management

Figure 3

The Program

Lines 10 through 220 contain the program header, initialize the program, print instructions, and print the menu of functions available. The multiple branch statement in line 230 sends the pointer to the proper section of the program as selected from the menu.

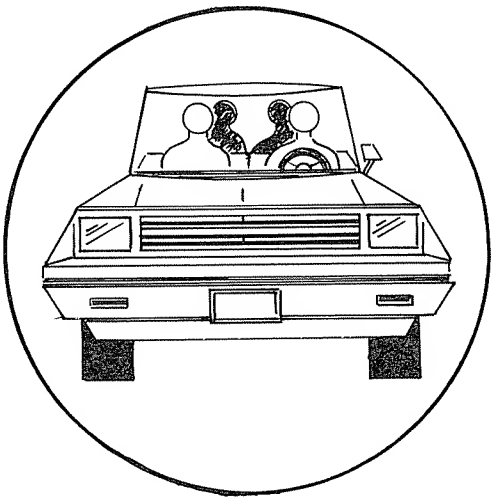
Lines 300 through 430 accomplish the function of building or adding to a file of data. Each line of data is prompted by the program, and the operator need only answer each prompt to build a file. Lines 325, 335, 345, and 355 set the size of each line. The string sizes of name, street, city, and zip are set at 30 characters each, while the telephone number has a field of 15 characters and zone is left at either one or two characters, depending on the zone entered. This field sizing is done by adding null characters up to the field size.

The reason behind all this string manipulation is to ensure that each data element such as name, address, city, etc., starts at the same position in the array containing the N$ variables. This must be done so that the portion of the program that breaks N$ back into the various fields knows where to start looking for each data element.

Lines 500 through 760 allow changes to be made to data already in the file. Here again each string is set to size and concatenated after the correction is made. Lines 800 through 940 are used to establish and print out the

desired relationship between the zones of the employees in the file. The subroutine in lines 3000 through 3060 is used to break down the concatenated string N$ back into the various elements represented by N1$, N2$, N3$, N4$, and N5$. The actual printing of the data elements contained in strings N1$ through N5$ is accomplished by the subroutine in lines 4000 through 4020.

First choice matching—where the zone is the same as that for the person being matched—is done in lines 5000 through 5050. The actual matching of zones is in line 5020. This line also prevents the printing of the person being matched as a choice. The subroutine in lines 6000 through 6880 selects the second-choice matches. A name is selected as a second choice if his or her zone is adjacent to the zone of the person being matched.



Using the Program

In order to ensure the least complicated operation, the program is written in the "tutorial mode," that is, the operator need only answer the questions displayed by the computer in order to run all of the functions.

Function 1—This function is used for initial entry of data into the file and is a simple matter of entering the required information in response to the questions: NAME?, STREET ADDRESS?, CITY & ZIP?, TELEPHONE NO.?, and ZONE?. The information is entered directly into the computer from the questionnaires returned by the employees and need not be in any particular order. The file is closed upon answering NO to the question DO YOU HAVE ANOTHER NAME TO ENTER? or upon reaching 80 names in the file.

Function 2—Since most operators are not trained typists, I felt it was necessary to allow a simple means of correcting inaccurate data. Entering the name of the person for whom the data is to be corrected will result in each line of data being displayed—one at a time—exactly as it appears on the file. If that line is to be changed, the correct information is typed in. If no correction is to be made to that line, the operator may move on to the next line simply by pressing ENTER. After all corrections have been completed, the entire corrected file is displayed.

Function 3—This function prints out the name, address, and telephone number of the person for whom a match is sought, followed by short instructions, then a list of first choices, and then a list of second choices. The first choices are those employees that have the same zone as the person for whom a match is being sought, while second choices are those who live in a zone which is adjacent. By studying the zoned map and comparing it to the chart in Table 1, you will see this relationship.

Functions 4 and 5—These functions are straight-ahead applications for saving and recalling data to and from a cassette file. As explained above, this process is shorter than usual due to the fact that only one string is transferred to the cassette file.

Zone	Zones that are adjacent
1	2,5,6
2	1,3,5,6,7
3	2,3,6,7,8
4	3,7,8
5	1,2,6,9,10
6	1,2,3,5,7,9,10,11
7	2,3,4,6,8,10,11,12
8	3,4,7,11,12
9	5,6,10,13,14
10	5,6,7,9,11,13,14,15
11	6,7,8,10,12,14,15,16
12	7,8,11,15,16
13	9,10,14
14	9,10,11,13,15
15	10,11,12,14,16
16	11,12,15

Table 1. *Relationship of zones used in selecting second-choice matches*

Program Listing

```
10 :
   ' **       CAR POOLING       **
20 :
   ' **    WALTER K. MCCAHAN    **
30 :
   ' **                         **
40 :
   ' **      TRS-80  LEVEL II   **
50 CLEAR 8800:
   DIM N$(81)
60 CLS :
   PRINT :
   PRINT :
   PRINT "THIS PROGRAM MATCHES RIDES THAT ARE AVAILIBLE FOR CAR POO
   LING":
   PRINT
70 PRINT "THE OUTPUT OF THE PROGRAM WILL BE THE NAME AND ADDRESS"
80 PRINT "OF THE PERSON FOR WHOM A MATCH IS SOUGHT FOLLOWED BY A"
90 PRINT "LIST OF FIRST CHOICES THEN A LIST OF SECOND CHOICES,"
100 PRINT "AND A LIST OF THIRD CHOICES. THE PERSON SHOULD BE"
110 PRINT "GIVEN THE LIST ALONG WITH THE INSTRUCTIONS TO CONTACT THE
   "
120 PRINT "LISTED PEOPLE, IN ORDER, AND ARRANGE A CAR POOL."
130 PRINT :
    PRINT :
    PRINT :
    PRINT
140 INPUT "          TO PROCEED PRESS ENTER";X
150 CLS :
    PRINT :
    PRINT :
    PRINT "THE FOLLOWING FUNCTIONS ARE AVAILIBLE:"
160 PRINT "      1. ENTER OR ADD TO LIST OF NAMES
170 PRINT "      2. MAKE CORRECTIONS TO AN EXISTING NAME
180 PRINT "      3. FIND A MATCH
190 PRINT "      4. SAVE A FILE ONTO CASSETTE
200 PRINT "      5. LOAD A FILE FROM CASSETTE
210 PRINT :
    PRINT
220 INPUT "ENTER THE NUMBER OF YOUR CHOICE";X
230 ON X GOTO 300,500,800,1200,1400
300 CLS :
    PRINT :
    PRINT :
    PRINT :
    INPUT "ENTER THE NUMBER OF NAMES ALREADY ON THE FILE";I
310 CLS :
    PRINT "ENTER THE PROPER INFORMATION IN RESPONSE TO EACH PROMPT":
    PRINT
315 FOR I = I + 1 TO 80
320  INPUT "NAME";N1$:
     PRINT
325  M1 = 30 - LEN(N1$):
     M$ = STRING$(M1," "):
     N1$ = N1$ + M$
330  INPUT "STREET & ADDRESS";N2$:
     PRINT
335  M2 = 30 - LEN(N2$):
     M2$ = STRING$(M2," "):
     N2$ = N2$ + M2$
340  INPUT "CITY & ZIP";N3$:
     PRINT
345  M3 = 30 - LEN(N3$):
     M3$ = STRING$(M3," "):
     N3$ = N3$ + M3$
350  INPUT "TELEPHONE NUMBER";N4$:
     PRINT
355  M4 = 15 - LEN(N4$):
     M4$ = STRING$(M4," "):
```

Program continued

```
     N4$ = N4$ + M4$
360  INPUT "ZONE";N5$:
     PRINT
370  PRINT :
     INPUT "DO YOU HAVE ANOTHER NAME TO ENTER";Q$
380  IF Q$ = "NO" GOTO 410
390  N$(I) = N1$ + N2$ + N3$ + N4$ + N5$
400  CLS :
     NEXT I
410 N$(I) = N1$ + N2$ + N3$ + N4$ + N5$
420 P1 = I
430 GOTO 150
500 :
    ' **   CORRECTION SUBROUTINE    **
510 CLS :
    PRINT :
    PRINT
520 INPUT "ENTER NAME FOR WHICH CORRECTION IS TO BE MADE";S$
522 POKE 16422,88:
    POKE 16423,4
525 M1 = 30 - LEN(S$):
    M$ = STRING$(M1," "):
    S$ = S$ + M$
530 FOR I = 1 TO P1:
     IF LEFT$(N$(I),30) = S$ GOTO 570
535  NEXT I
540 CLS :
    PRINT S$;" NOT FOUND ON FILE"
550 PRINT :
    INPUT "WANT TO TRY AGAIN";Q$
555 IF Q$ = "NO" GOTO 150
560 GOTO 500
570 GOSUB 3000
580 CLS :
    PRINT "EACH LINE OF INFORMATION WILL BE DISPLAYED AS"
590 PRINT "IT APPEARS ON THE FILE. IF IT IS TO BE CORRECTED"
600 PRINT "THE CORRECT INFORMATION SHOULD BE TYPED IN. IF"
610 PRINT "NO CORRECTION IS TO BE MADE TO THAT LINE PRESS ENTER."
620 PRINT :
    PRINT N1$
630 INPUT N1$
635 M1 = 30 - LEN(N1$):
    M$ = STRING$(M1," "):
    N1$ = N1$ + M$
640 PRINT N2$
650 INPUT N2$
655 M2 = 30 - LEN(N2$):
    M2$ = STRING$(M2," "):
    N2$ = N2$ + M2$
660 PRINT N3$
670 INPUT N3$
675 M3 = 30 - LEN(N3$):
    M3$ = STRING$(M3," "):
    N3$ = N3$ + M3$
680 PRINT N4$
690 INPUT N4$
695 M4 = 15 - LEN(N4$):
    M4$ = STRING$(M4," "):
    N4$ = N4$ + M4$
700 PRINT N5$
710 INPUT N5$
715 N$(I) = N1$ + N2$ + N3$ + N4$ + N5$
720 CLS :
    PRINT "FOLLOWING IS THE CORRECTED FILE"
730 GOSUB 4000
735 POKE 16422,141:
    POKE 16423,5
740 PRINT :
    INPUT "DO YOU HAVE ANOTHER CORRECTION TO MAKE";Q$
750 IF Q$ = "YES" GOTO 500
760 GOTO 150
```

```
800 :
    ' **     MATCH A SPECIFIC PERSON     **
810 CLS :
    PRINT :
    PRINT :
    INPUT "ENTER NAME FOR WHICH A MATCH IS SOUGHT";S$
815 M1 = 30 - LEN(S$):
    M$ = STRING$(M1," "):
    S$ = S$ + M$
820 FOR I = 1 TO P1:
     IF S$ = LEFT$(N$(I),30) GOTO 840
825  NEXT I
830 PRINT S$;" NOT FOUND ON FILE":
    GOTO 920
840 GOSUB 3000
845 N9$ = N5$:
    N5 = VAL(N5$)
850 GOSUB 4000
860 LPRINT :
    LPRINT "     FOLLOWING IS A LIST OF POSSIBLE RIDERS FOR THE"
870 LPRINT "PERSON NAMED ABOVE. THE RIDERS LISTED UNDER FIRST"
880 LPRINT "CHOICE ARE MOST DESIRABLE, BUT THE SECOND CHOICES"
890 LPRINT "ARE ALSO BENEFICIAL."
900 GOSUB 5000
910 GOSUB 6000
920 PRINT :
    PRINT :
    INPUT "DO YOU HAVE ANOTHER NAME TO MATCH";Q$
930 IF Q$ = "YES" GOTO 800
940 GOTO 150
1200 :
    ' **     SAVE ONTO CASSETTE     **
1210 CLS :
    PRINT :
    PRINT :
    PRINT
1220 INPUT "WHEN CASSETTE IS READY PRESS ENTER";X
1230 CLS :
    PRINT :
    PRINT :
    PRINT
1240 PRINT "NOW RECORDING ONTO CASSETTE"
1250 PRINT # - 1,P1
1260 FOR I = 1 TO P1
1270  PRINT # - 1,N$(I)
1280  NEXT I
1290 CLS :
    PRINT :
    PRINT :
    PRINT
1300 INPUT "RECORDING COMPLETE. PRESS ENTER TO PROCEED.";X
1310 GOTO 150
1400 :
    ' **     LOAD FROM CASSETTE     **
1410 CLS :
    PRINT :
    PRINT :
    PRINT
1420 INPUT "WHEN CASSETTE IS READY PRESS ENTER";X
1430 CLS
1440 PRINT :
    PRINT :
    PRINT :
    PRINT "NOW LOADING FROM CASSETTE"
1450 INPUT # - 1,P1
1460 FOR I = 1 TO P1
1470  INPUT # - 1,N$(I)
1480  NEXT I
1490 CLS :
    PRINT :
```

Program continued

```
     PRINT :
     PRINT "LOADING NOW COMPLETE, ";P1;" ITEMS ON THE FILE"
1500 PRINT :
     PRINT :
     INPUT "PRESS ENTER TO PROCEED";X
1510 GOTO 150
3000 :
     ' **     TO BREAK DOWN STRINGS     **
3010 N1$ = LEFT$(N$(I),30)
3020 N2$ = MID$(N$(I),31,30)
3030 N3$ = MID$(N$(I),61,30)
3040 N4$ = MID$(N$(I),91,8)
3050 N5$ = RIGHT$(N$(I),2)
3055 SW = 1
3060 RETURN
4000 :
     ' **      TO PRINT A RECORD     **
4010 LPRINT :
     LPRINT TAB(10)N1$:
     LPRINT TAB(10)N2$:
     LPRINT TAB(10)N3$:
     LPRINT TAB(10)N4$:
     LPRINT TAB(10)N5$
4020 RETURN
5000 :
     ' **     TO SELECT FIRST CHOICES     **
5010 FOR I = 1 TO P1 + 1
5015  GOSUB 3000
5020  IF N9$ = N5$ AND S$ < > N1$ GOTO 5030 :
       ELSE
        5040
5030  GOSUB 4000
5035  SW = 0
5040  NEXT I
5050 RETURN
6000 :
     ' **     TO SELECT SECOND CHOICES     **
6005 LPRINT :
     LPRINT :
     LPRINT "         SECOND CHOICES":
     LPRINT
6010 FOR I = 1 TO P1
6020  R = VAL( RIGHT$(N$(I),2))
6030  IF R = 2 AND N5 = 1 GOSUB 3000
6040  IF R = 6 AND NN5 = 1 GOSUB 3000
6050  IF R = 5 AND N5 = 1 GOSUB 3000
6060  IF R = 1 AND N5 = 2 GOSUB 3000
6070  IF R = 3 AND N5 = 2 GOSUB 3000
6080  IF R = 5 AND N5 = 2 GOSUB 3000
6090  IF R = 6 AND N5 = 2 GOSUB 3000
6100  IF R = 7 AND N5 = 2 GOSUB 3000
6110  IF R = 2 AND N5 = 3 GOSUB 3000
6120  IF R = 4 AND N5 = 3 GOSUB 3000
6130  IF R = 6 AND N5 = 3 GOSUB 3000
6140  IF R = 7 AND N5 = 3 GOSUB 3000
6150  IF R = 8 AND N5 = 3 GOSUB 3000
6160  IF R = 3 AND N5 = 4 GOSUB 3000
6170  IF R = 7 AND N5 = 4 GOSUB 3000
6180  IF R = 8 AND N5 = 4 GOSUB 3000
6190  IF R = 1 AND N5 = 5 GOSUB 3000
6200  IF R = 2 AND N5 = 5 GOSUB 3000
6210  IF R = 6 AND N5 = 5 GOSUB 3000
6220  IF R = 9 AND N5 = 5 GOSUB 3000
6230  IF R = 10 AND N5 = 5 GOSUB 3000
6240  IF R = 1 AND N5 = 6 GOSUB 3000
6250  IF R = 2 AND N5 = 6 GOSUB 3000
6260  IF R = 3 AND N5 = 6 GOSUB 3000
6270  IF R = 5 AND N5 = 6 GOSUB 3000
6280  IF R = 7 AND N5 = 6 GOSUB 3000
6290  IF R = 9 AND N5 = 6 GOSUB 3000
```

```
6300  IF R = 10 AND N5 = 6 GOSUB 3000
6310  IF R = 11 AND N5 = 6 GOSUB 3000
6320  IF R = 2 AND N5 = 7 GOSUB 3000
6330  IF R = 3 AND N5 = 7 GOSUB 3000
6340  IF R = 4 AND N5 = 7 GOSUB 3000
6350  IF R = 6 AND N5 = 7 GOSUB 3000
6360  IF R = 8 AND N5 = 7 GOSUB 3000
6370  IF R = 10 AND N5 = 7 GOSUB 3000
6380  IF R = 11 AND N5 = 7 GOSUB 3000
6390  IF R = 12 AND N5 = 7 GOSUB 3000
6400  IF R = 3 AND N5 = 8 GOSUB 3000
6410  IF R = 4 AND N5 = 8 GOSUB 3000
6420  IF R = 7 AND N5 = 8 GOSUB 3000
6430  IF R = 11 AND N5 = 8 GOSUB 3000
6440  IF R = 12 AND N5 = 8 GOSUB 3000
6445  IF R = 5 AND N5 = 9 GOSUB 3000
6450  IF R = 6 AND N5 = 9 GOSUB 3000
6460  IF R = 10 AND N5 = 9 GOSUB 3000
6470  IF R = 13 AND N5 = 9 GOSUB 3000
6480  IF R = 14 AND N5 = 9 GOSUB 3000
6490  IF R = 5 AND N5 = 10 GOSUB 3000
6500  IF R = 6 AND N5 = 10 GOSUB 3000
6510  IF R = 7 AND N5 = 10 GOSUB 3000
6520  IF R = 9 AND N5 = 10 GOSUB 3000
6530  IF R = 11 AND N5 = 10 GOSUB 3000
6540  IF R = 13 AND N5 = 10 GOSUB 3000
6550  IF R = 14 AND N5 = 10 GOSUB 3000
6560  IF R = 15 AND N5 = 10 GOSUB 3000
6570  IF R = 6 AND N5 = 11 GOSUB 3000
6580  IF R = 7 AND N5 = 11 GOSUB 3000
6590  IF R = 8 AND N5 = 11 GOSUB 3000
6600  IF R = 10 AND N5 = 11 GOSUB 3000
6610  IF R = 12 AND N5 = 11 GOSUB 3000
6620  IF R = 14 AND N5 = 11 GOSUB 3000
6630  IF R = 15 AND N5 = 11 GOSUB 3000
6640  IF R = 16 AND N5 = 11 GOSUB 3000
6650  IF R = 7 AND N5 = 12 GOSUB 3000
6660  IF R = 8 AND N5 = 12 GOSUB 3000
6670  IF R = 11 AND N5 = 12 GOSUB 3000
6680  IF R = 15 AND N5 = 12 GOSUB 3000
6690  IF R = 16 AND N5 = 12 GOSUB 3000
6700  IF R = 9 AND N5 = 13 GOSUB 3000
6710  IF R = 10 AND N5 = 13 GOSUB 3000
6720  IF R = 14 AND N5 = 13 GOSUB 3000
6730  IF R = 9 AND N5 = 14 GOSUB 3000
6740  IF R = 10 AND N5 = 14 GOSUB 3000
6750  IF R = 11 AND N5 = 14 GOSUB 3000
6760  IF R = 13 AND N5 = 14 GOSUB 3000
6770  IF R = 15 AND N5 = 14 GOSUB 3000
6780  IF R = 10 AND N5 = 15 GOSUB 3000
6790  IF R = 11 AND N5 = 15 GOSUB 3000
6800  IF R = 12 AND N5 = 15 GOSUB 3000
6810  IF R = 14 AND N5 = 15 GOSUB 3000
6820  IF R = 16 AND N5 = 15 GOSUB 3000
6830  IF R = 11 AND N5 = 16 GOSUB 3000
6840  IF R = 12 AND N5 = 16 GOSUB 3000
6850  IF R = 15 AND N5 = 16 GOSUB 3000
6860  IF SW = 1 AND S$ < > N1$ GOSUB 4000
6865  SW = 0
6870  NEXT I
6880 RETURN
```

HOME APPLICATIONS

Doctor Your Records

by Wilbur A. Muehlig, M.D.

When I was practicing medicine, my secretary kept a "daybook," entering each payment made by a patient, adding totals for the day, the month, and finally, the year.

When I left private practice, I continued the same general method of keeping track of income checks, but my usual practice was to let the arithmetic go until the end of the year when my taxes were due. At this time totaling my income was a rather discouraging job, but ideally suited to a computer.

This bookkeeping program should be of interest to anyone owning a few stocks and bonds. It is written with the retired person in mind, and therefore includes Social Security income and annuity payments.

The program, which handles up to 35 entries a month, accepts eight types of entries: earned income, dividends, interest, annuity funds, tax-free interest, Social Security, life insurance dividends, and miscellaneous items. The first four are taxable, the next three untaxable, and the last is for your records only.

The program is written in Level II BASIC for a TRS-80 with 32K and a single disk drive, but it can be easily modified to use a cassette. The printout has a line length of 61 characters, but can be compressed to fit into 40 columns. Though the program can be used without a printer, much of its benefit is lost.

First Steps

Choose MAKE A FILE ENTRY (number three in Table 1) when first using the program. Enter the day of the month and the source of the income; choose the type of entry, the amount (Table 2), and press ENTER.

When you have finished making your entries, save them to disk using the first menu option and the disk option: SAVE MONTH'S FILE TO DISK (one in Table 1). Before adding more entries for the same month, load the previous ones with disk option two.

You can use menu options four (CORRECT AN ENTRY), five (DELETE AN ENTRY), or six (SEE A PRINTOUT ON VIDEO) at any time while making these entries. Option four is also useful for checking the income type you have chosen, since it is not printed by either the video or hard copy printouts.

After the month's entries are complete, disk option four loads the yearly totals from the previous month. Menu option two totals each type of income

for the month and the year and then displays the results. You can use disk option three to save the totals, menu option six for a complete video printout, and seven for hard copy (See Table 3 for sample).

```
WHAT DO YOU WANT TO DO?

    1—DISK OPERATIONS
    2—COMPUTATIONS
    3—MAKE A FILE ENTRY
    4—CORRECT AN ENTRY
    5—DELETE AN ENTRY
    6—SEE A PRINTOUT ON VIDEO
    7—MAKE A HARD COPY
    8—LEAVE THE PROGRAM

CHOOSE? 1


WHAT DO YOU WANT TO DO?

    1—SAVE MONTH'S FILE TO DISK
    2—LOAD MONTH'S FILE FROM DISK
    3—SAVE YEARLY TOTALS TO DATE
    4—LOAD YEARLY TOTALS
    5—RETURN TO MENU

CHOOSE? 1
```

Table 1

Changes

Changing to cassette data files will require alterations between lines 240 and 370. Sequential disk files are quite similar to cassette files, and use of the *Level II BASIC Reference Manual* should make this change simple.

Taxable items can be changed to nontaxable or vice versa in lines 7000-7070 and 6099-6820. Note that L( ) and R( ) collect the taxable and nontaxable items for printing in columns. Refer to the list of variables (Table 4) as necessary.

Miscellaneous entries include loan repayments, refunds, and money from the sale of stocks and bonds. The program doesn't handle capital gains because of their special requirements, but you can type the data on the back of the monthly pages.

This program considers life insurance dividends as nontaxable income, al-

```
  1
TO END FILE ENTRIES, TYPE 99

DAY OF MONTH? 5

SOURCE OF INCOME (NOT OVER 24 SPACES)
? AT & T
WHAT TYPE OF ENTRY?
1—EARNED INCOME     5—TAX FREE INTEREST
2—DIVIDEND          6—SOCIAL SECURITY
3—INTEREST          7—LIFE INSURANCE DIVIDEND
4—ANNUITY           8—MISCELLANEOUS ITEM

CHOOSE? 2

AMOUNT OF DIVIDEND? 345.67_

1  DATE 5   AT & T DIVIDEND $ 345.67

IF THE ABOVE IS INCORRECT, TYPE 1. IF THE PART THEN DISPLAYED IS COR-
RECT, PRESS ENTER, OTHERWISE TYPE THE CORRECTION.

NOTE: IF THE ERROR IS IN THE TYPE OF INCOME, DELETE THE WHOLE ITEM
AND REENTER.

IF THE ABOVE IS CORRECT, TYPE 2 TO GO ON?_
```

Table 2

though they are actually premium refunds. You can change this by omitting " +FS" from line 7060.

Annuity payments can also be modified to record a second earned income by changing the entry name in lines 660, 694, 1310, 1510, and 6410.

The printout is for five-and-one-half by eight-and-one-half-inch paper at 12 characters per inch. The printout itself is 54 characters wide with a seven-character margin.

You can narrow the printout by as much as 14 characters by reducing the Source of Income heading from 24 to 10 and removing the same number of spaces from between the percentage signs in lines 1440 and 1450. Abbreviate other terms such as Total Earned Income (line 1485), and decrease the first number in STRING$(54,45) (lines 1420 and 1460).

To use a 40-column printer, get rid of the margin by removing TAB(7) after each LPRINT.

If you decrease the hard copy width and want your video printout to match, make the same changes in lines 1200 to 1390.

About the Program

The program asks for the year and month and automatically generates data files. In any month but January, the yearly total file will be from the preceding month. Do the computations in January without loading the yearly totals.

The program will allow you to return to the menu if you choose a wrong number. If you press the BREAK key and get into BASIC, type GOTO 100 to get back to the menu without losing data files.

Line 240 saves the original file if you accidentally choose SAVE MONTH'S FILE TO DISK instead of LOAD MONTH'S FILE FROM DISK. Without some such safeguard the program could file a string of zeros and erase the original file. Line 299 gives a similar safeguard for SAVE YEARLY TOTALS TO DATE.

An ON ERROR GOTO prevents data loss if you choose LOAD MONTH'S FILE FROM DISK when there is no such file or pick LOAD YEARLY TOTALS.

In regard to disk space, none of my summary data has taken up more than one gran. Since the program requires nine grans, the total disk space, including the program, for a year comes to 32 grans and fits nicely onto one disk. Using hard copy, the same disk can be used year after year.

##	DAY	SOURCE OF INCOME	TAXABLE	NONTAX.
1	1	TIAA	$ 225.68	
2	1	IA PS	$ 153.00	
3	3	NW PS	$ 212.50	
4	3	SS (PD TO BANK)		$ 418.10
5	3	MASS MUT DIV		$ 33.15
6	5	GAS SERV	$ 160.00	
7	10	NE INV TRUST	$ 178.52	
8	10	TEXACO	$ 50.00	
9	15	OMAHO NAT'L CORP	$ 225.00	
10	15	MUNI INV TRUST		$ 58.10
11	15	TESORO	$ 108.00	
12	17	CORP INC FUND	$ 35.05	
13	21	KC P & L	$ 256.00	
14	29	JAPAN FUND	$ 15.00	

TOTALS FOR MAR 1979	MONTH	YEAR TO DATE
TOTAL EARNED INCOME	$ 0.00	$ 0.00
TOTAL DIVIDENDS	$ 1,393.07	$ 1,740.12

Table 3. *Income Ledger for March 1979*

Y$	Year
V$	Yearly file name
M( )	String, name of month
H	Number of month
DA( )	Day of month
S( )	String, source of income
E( ),ET,ES	Earned income, monthly total, yearly total
D( ),DT,DS	Dividends, monthly total, yearly total
I( ),IT,IS	Interest, monthly total, yearly total
A( ),AT,AS	Annuity, monthly total, yearly total
T( ),TT,TS	Tax free interest, monthly total, yearly total
G( ),GT,GS	Social Security, monthly total, yearly total
F( ),FT,FS	Life insurance dividend, monthly total, yearly total
C( )	Miscellaneous items
L( )	All taxable items
LS	Total taxable income to date
R( )	All nontaxable items
RS	Total nontaxable income to date
Q	Total income to date
U$	Formatter
X	File counter
B,J,K,Z	Counters and null
P1	Number of items plus one (99 to end entries)

Table 4. *Variables for Ledger Program*

Program Listing

```
  1 REM  * PERSONAL INCOME LEDGER PROGRAM *
  2 REM  * WILBUR A. MUEHLIG, M.D. *
  3 REM  * 726 N. 91 PLAZA, APT. 305,
  4 REM  * OMAHA, NE 68114
 11 REM  * OCT. 1979.
 12 CLS
 13 PRINT :
    PRINT :
    PRINT :
    PRINT :
    PRINT :
    PRINT :
    PRINT
 15 PRINT TAB(23)"INCOME LEDGER PROGRAM":
    PRINT :
    PRINT :
    PRINT
 20 INPUT "DO YOU WANT INSTRUCTIONS FOR USE (Y/N)";Z$
 30 IF Z$ = "Y" GOTO 4000
 40 CLEAR 500:
    DEFDBL E,D,I,A,T,G,L,C,F,R,Q:
    DEFINT B,J,K,H,X,Z,P:
    P1 = 1
 45 DEFSTR S,M,V,Y,U
 50 DIM S(35),E(25),DA(35),D(25),I(25),A(25),T(25),G(25),L(35),C(25)
    ,F(35),R(35),M(12)
 52 DATA JAN,FEB,MAR,APR,MAY,JUN,JUL,AUG,SEP,OCT,NOV,DEC
 53 FOR H = 1 TO 12:
     READ M$(H):
     NEXT
 55 CLS :
    PRINT :
    PRINT :
    PRINT :
    PRINT
 60 INPUT "YEAR OF THIS REPORT (####)";Y$
 80 PRINT :
    INPUT "NUMBER OF THIS MONTH (1 TO 12)";H
100 CLS :
    PRINT
110 PRINT "
WHAT DO YOU WANT TO DO?
1-DISK OPERATIONS      2-COMPUTATIONS
3-MAKE A FILE ENTRY    4-CORRECT AN ENTRY
5-DELETE AN ENTRY      6-SEE A PRINTOUT ON VIDEO
7-MAKE A HARD COPY     8-LEAVE THE PROGRAM"
114 PRINT :
    INPUT "CHOOSE";J
120 ON J GOTO 200,1262,600,800,1000,1225,1400,1600
200 CLS :
     PRINT :
     PRINT :
     PRINT :
     PRINT :
     PRINT "
WHAT DO YOU WANT TO DO?
1-SAVE MONTH'S FILE TO DISK     2-LOAD MONTH'S FILE FROM DISK
3-SAVE YEARLY TOTALS TO DATE    4-LOAD YEARLY TOTALS
5-RETURN TO MENU"
220 PRINT :
    INPUT "CHOOSE";J
230 ON J GOTO 240,270,299,349,235
235 GOTO 100
```

Program continued

```
240 IF S(1) = ""
     THEN
      PRINT "FILE EMPTY.":
      INPUT "PRESS ENTER";Z$:
      GOTO 200
241 OPEN "O",1,M(H)
242 PRINT "SAVING MONTHLY FILE TO DISK....."
245 PRINT #1,P1
250 FOR X = 1 TO P1 - 1
255  PRINT #1,S(X):
     PRINT #1,X;DA(X);E(X);D(X);I(X);A(X);T(X);G(X);F(X);C(X)
260  NEXT :
    CLOSE
265 PRINT :
    PRINT "MONTHLY FILE SAVED TO DISK":
    FOR Z = 1 TO 500:
     NEXT :
    GOTO 100
270 ON ERROR GOTO 1800
272 OPEN "I",1,M(H)
275 INPUT #1,P1
280 FOR X = 1 TO P1 - 1
285  INPUT #1,S(X):
     INPUT #1,X,DA(X),E(X),D(X),I(X),A(X),T(X),G(X),F(X),C(X)
287  PRINT X;S(X)
290  NEXT :
    CLOSE
291 FOR X = 1 TO P1 - 1:
     L(X) = E(X) + D(X) + I(X) + A(X):
     R(X) = T(X) + G(X) + F(X) + C(X):
     NEXT
295 PRINT :
    PRINT "MONTHLY FILE LOADED.";:
    INPUT "    PRESS ENTER";Z$:
    GOTO 100
299 IF ES=0 AND DS=0 AND IS=0 AND AS=0 AND TS=0 AND GS
     =0 FS=0 AND CS=0
       THEN
         INPUT "FILE EMPTY.  PRESS ENTER";Z$ :
         GOTO 200
300 V$ = M(H) + Y$:
    OPEN "O",2,V$
305 PRINT :
    PRINT "SAVING YEARLY TOTALS TO DISK..."
320 PRINT #2,ES;DS;IS;AS;TS;GS;FS:
    CLOSE
330 GOTO 100
349 ON ERROR GOTO 1800
350 V$ = M(H - 1) + Y$
355 OPEN "I",2,V$
360 PRINT :
    PRINT "LOADING YEARLY TOTALS...."
370 INPUT #2,ES,DS,IS,AS,TS,GS,FS:
    CLOSE
380 GOTO 100
599 REM  * TO MAKE A FILE ENTRY *
600 CLS :
    PRINT :
    PRINT :
    PRINT
602 FOR X = P1 TO 50
610  CLS :
     PRINT :
     PRINT :
     PRINT
612  PRINT X
613  PRINT :
     PRINT "TO END FILE ENTRIES, TYPE 99"
615  PRINT :
     IF DA(X) < > 0 PRINT DA(X);" ";
```

```
620  INPUT "DAY OF MONTH";DA(X)
622  IF DA(X) = 99
      THEN
       P1 = X:
       GOTO 100
630  IF S(X) < > "" PRINT S(X);"   ";
640  PRINT :
     PRINT "SOURCE OF INCOME (NOT OVER 24 SPACES)":
     INPUT S(X)
645  IF LEN(S(X)) > 24 PRINT "TOO LONG. TRY AGAIN.":
     GOTO 640
660  PRINT "WHAT TYPE OF ENTRY?1-EARNED INCOME       5-TAX FREE INTE
     REST2-DIVIDEND            6-SOCIAL SECURITY3-INTEREST
       7-LIFE INSURANCE DIVIDEND4-ANNUITY           8-MISCELLANEOU
     S ITEM
662  ON ERROR GOTO 1700
665  PRINT :
     INPUT "CHOOSE";B
670  ON B GOTO 6100,6200,6300,6400,6500,6600,6700,6800
680  CLS :
     PRINT :
     PRINT
690  PRINT :
     PRINT X;" DATE";DA(X);"  ";S(X)"  ";
691  IF E(X) > 0 PRINT "EARNED INCOME   ";:
     PRINT USING "$###,##.##";E(X)
692  IF D(X) > 0 PRINT "DIVIDEND   ";:
     PRINT USING "$###,##.##";D(X)
693  IF I(X) > 0 PRINT "INTEREST   ";:
     PRINT USING "$###,##.##";I(X)
694  IF A(X) > 0 PRINT "ANNUITY   ";:
     PRINT USING "$###,##.##";A(X)
695  IF T(X) > 0 PRINT "TAX FREE INTEREST   ";:
     PRINT USING "$###,##.##";T(X)
696  IF G(X) > 0 PRINT "SOCIAL SECURITY   ";:
     PRINT USING "$###,##.##";G(X)
697  IF F(X) > 0 PRINT "LIFE INS. DIV.   ";:
     PRINT USING "$###,##.##";F(X)
698  IF C(X) > 0 PRINT "MISC. DEPOSIT   ";:
     PRINT USING "$###,##.##";C(X)
700  PRINT :
     PRINT "IF THE ABOVE IS INCORRECT, TYPE 1.  IF THE PART THEN DIS
     PLAYED"
705  PRINT "IS CORRECT, PRESS ENTER, OTHERWISE TYPE THE CORRECTION."
707  PRINT :
     PRINT "NOTE: IF THE ERROR IS IN THE TYPE OF INCOME, DELETE THE
     WHOLE"
708  PRINT "ITEM AND REENTER."
709  REM  * INCOME TYPE ERROR CAN CAUSE CONFUSION IF NOT DELETED *
710  PRINT :
     INPUT "IF THE ABOVE IS CORRECT, TYPE 2 TO GO ON";Z
720  IF Z = 1 GOTO 610
730  IF Z < > 2 GOTO 710
740  NEXT :
    GOTO 100
799 REM  * USES SAME CORRECTION SECTION AS FILE ENTRY *
800 CLS :
    PRINT :
    PRINT :
    PRINT :
    PRINT
810 PRINT "WHICH NUMBER DO YOU WISH TO CORRECT"
815 PRINT :
    PRINT "           (0 TO RETURN TO MENU)           ";:
    INPUT K
817 IF K = 0 GOTO 100
820 FOR X = 1 TO P1 - 1
830  IF K = X GOTO 680
840  NEXT
850 GOTO 100
```

Program continued

```
 999 REM  * DELETES ENTRY AND RENUMBERS *
1000 PRINT "WHICH NUMBER DO YOU WANT TO DELETE?"
1002 PRINT "      (0 TO RETURN TO MENU)            ";:
     INPUT X
1005 IF X = 0 GOTO 100
1010 DA(X) = 0:
     S(X) = "":
     E(X) = 0:
     D(X) = 0:
     I(X) = 0:
     A(X) = 0:
     T(X) = 0:
     G(X) = 0:
     F(X) = 0:
     C(X) = 0:
     L(X) = 0:
     R(X) = 0
1020 FOR X = X + 1 TO P1:
      DA(X - 1) = DA(X):
      S(X - 1) = S(X):
      E(X - 1) = E(X):
      D(X - 1) = D(X):
      I(X - 1) = I(X):
      A(X - 1) = A(X):
      T(X - 1) = T(X):
      G(X - 1) = G(X):
      F(X - 1) = F(X):
      C(X - 1) = C(X):
      L(X - 1) = L(X):
      R(X - 1) = R(X)
1025  NEXT
1030 P1 = P1 - 1
1040 PRINT :
     PRINT "ENTRY DELETED.":
     FOR Z = 1 TO 700:
      NEXT :
     GOTO 100
1200 CLS :
     PRINT TAB(16)"INCOME LEDGER FOR ";M(H);" ";Y$:
     PRINT
1210 PRINT "## DAY  SOURCE OF INCOME            TAXABLE   NONTAX."
1220 PRINT STRING$(54,45):
     RETURN
1225 GOSUB 1200 REM  * ALLOWS REPRINTING OF HEADING ON EACHPAGE OF VI
     DEO *
1230 FOR X = 1 TO P1 - 1
1235  IF X / 11 = INT(X / 11) PRINT :
      INPUT "PRESS ENTER TO CONTINUE";Z$:
      CLS :
      PRINT :
      GOSUB 1200
1240  IF R(X) = 0 PRINT USING "## ##  %                        %  $###,
      ##.##";X;DA(X);S(X);L(X)
1250  IF L(X) = 0 PRINT USING "## ##  %                          %
           $###,##.##";X;DA(X);S(X);R(X)
1260  NEXT :
     PRINT :
     INPUT "ENTER 1 TO RETURN TO MENU, 2 TO SEE YEARLY TOTALS.";Z:
     IF Z = 1 GOTO 100 :
      ELSE
       1279
1262 CLS :
     PRINT :
     PRINT :
     PRINT :
     PRINT "THE FOLLOWING COMPUTES INCOME TOTALS FOR THE MONTH AND FO
     R"
1263 PRINT "THE YEAR.  THE MONTH'S ENTRIES SHOULD BE COMPLETE, YOUR P
     RINTER"
1264 PRINT "READY, AND THE YEARLY TOTALS INPUT FROM DISK BEFORE CONTI
```

```
     NUING."
1265 PRINT :
     PRINT "TO RETURN TO MENU, TYPE 1, TO CONTINUE WITH THE COMPUTATI
     ON,"
1266 PRINT "TYPE 2.":
     PRINT :
     INPUT Z
1267 IF Z = 1 GOTO 100
1268 IF Z < > 2 GOTO 1262
1270 GOSUB 7000
1279 CLS
1280 PRINT "   TOTALS FOR ";M(H);" ";Y$;"         MONTH       YEAR TO D
     ATE"
1282 PRINT
1285 PRINT "TOTAL EARNED INCOME          ";:
     PRINT USING "$####,###.##  $####,###.##";ET,ES
1290 PRINT "TOTAL DIVIDENDS              ";:
     PRINT USING "$####,###.##  $####,###.##";DT,DS
1300 PRINT "TOTAL INTEREST               ";:
     PRINT USING "$####,###.##  $####,###.##";IT,IS
1310 PRINT "TOTAL ANNUITY FUNDS          ";:
     PRINT USING "$####,###.##  $####,###.##";AT,AS
1320 PRINT "TOTAL TAX FREE INTEREST      ";:
     PRINT USING "$####,###.##  $####,###.##";TT,TS
1330 PRINT "SOCIAL SECURITY              ";:
     PRINT USING "$####,###.##  $####,###.##";GT,GS
1340 PRINT "TOTAL LIFE INS. DIV.         ";:
     PRINT USING "$####,###.##  $####,###.##";FT,FS
1355 PRINT :
     U$ = "$####,###.##"
1360 PRINT "TOTAL TAXABLE INCOME TO DATE          ";:
     PRINT USING U$;LS
1370 PRINT "TOTAL NONTAX. INCOME TO DATE          ";:
     PRINT USING U$;RS
1380 PRINT "TOTAL INCOME TO DATE                  ";:
     PRINT USING U$;Q
1390 PRINT :
     INPUT "PRESS ENTER TO CONTINUE";Z$:
     GOTO 100
1400 CLS :
     PRINT :
     PRINT :
     PRINT :
     PRINT :
     PRINT :
     PRINT "IF YOUR PRINTER ISN'T READY, ENTER 1 TO RETURN TO MENU,"
1402 PRINT :
     INPUT "          OTHERWISE ENTER 2.";Z:
     IF Z = 1 GOTO 100
1405 CLS :
     LPRINT TAB(20)"INCOME LEDGER FOR ";M(H);" ";Y$:
     LPRINT ""
1410 LPRINT TAB(7)"## DAY  SOURCE OF INCOME            TAXABLE   NONT
     AX."
1420 LPRINT TAB(7) STRING$(54,45)
1430 FOR X = 1 TO P1 - 1
1440  IF R(X) = 0 LPRINT TAB(7) USING "## ##  %
      %  $###,##.##";X;DA(X);S(X);L(X)
1450  IF L(X) = 0 LPRINT TAB(7) USING "## ##  %
        %          $###,##.##";X;DA(X);S(X);R(X)
1460  NEXT :
     LPRINT TAB(7) STRING$(54,45):
     LPRINT ""
1480 LPRINT TAB(7)"   TOTALS FOR ";M(H);" ";Y$;"         MONTH       YE
     AR TO DATE"
1482 LPRINT ""
1485 LPRINT TAB(7)"TOTAL EARNED INCOME          ";:
     LPRINT TAB(7) USING "$####,###.##  $####,###.##";ET,ES
1490 LPRINT TAB(7)"TOTAL DIVIDENDS              ";:
     LPRINT TAB(7) USING "$####,###.##  $####,###.##";DT,DS
```

Program continued

```
1500 LPRINT TAB(7)"TOTAL INTEREST                ";:
     LPRINT TAB(7) USING "$####,###.##  $####,###.##";IT,IS
1510 LPRINT TAB(7)"TOTAL ANNUITY FUNDS           ";:
     LPRINT TAB(7) USING "$####,###.##  $####,###.##";AT,AS
1520 LPRINT TAB(7)"TOTAL TAX FREE INTEREST       ";:
     LPRINT TAB(7) USING "$####,###.##  $####,###.##";TT,TS
1530 LPRINT TAB(7)"SOCIAL SECURITY               ";:
     LPRINT TAB(7) USING "$####,###.##  $####,###.##";GT,GS
1540 LPRINT TAB(7)"TOTAL LIFE INS. DIV.          ";:
     LPRINT TAB(7) USING "$####,###.##  $####,###.##";FT,FS
1555 U$ = "$####,###.##"
1557 LPRINT ""
1560 LPRINT TAB(7)"TOTAL TAXABLE INCOME TO DATE          ";:
     LPRINT TAB(7) USING U$;LS
1570 LPRINT TAB(7)"TOTAL NONTAX. INCOME TO DATE          ";:
     LPRINT TAB(7) USING U$;RS
1580 LPRINT TAB(7)"TOTAL INCOME TO DATE                  ";:
     LPRINT TAB(7) USING U$;Q
1590 PRINT :
     INPUT "PRESS ENTER TO CONTINUE";Z$:
     GOTO 100
1600 CLS :
     PRINT :
     PRINT :
     PRINT :
     PRINT :
     PRINT :
     PRINT :
     PRINT :
     INPUT "HAVE YOU SAVED FILES TO DISK (Y/N)";Z$
1610 IF Z$ = "N" GOTO 100
1620 PRINT :
     PRINT "OK. SO LONG!":
     END
1700 PRINT :
     INPUT "ENTRY ERROR.  PRESS ENTER AND TRY AGAIN.";Z$:
     GOTO 665
1800 PRINT :
     INPUT "NO FILE ON DISK.  PRESS ENTER.";Z$:
     GOTO 100
4000 CLS :
     PRINT :
     PRINT
4010 PRINT "                    PERSONAL INCOME LEDGER":
     PRINT
4020 PRINT "1) DISK FILES ARE NAMED AUTOMATICALLY.  THE MONTHLY FILES
      ARE"
4030 PRINT "JAN, FEB, ETC.  THE YEARLY TOTAL FILES ARE JAN1979, FEB19
     79,"
4040 PRINT "ETC., AND ACCUMULATE THE FIGURES FOR THE YEAR TO DATE."
4060 PRINT
4090 PRINT "2) MAKE ENTRIES FOR THE MONTH.  SAVE TO DISK WITH DISK OP
     TION"
4100 PRINT "#1.  BEFORE ADDING ENTRIES FOR THE SAME MONTH, USE DISK O
     PTION"
4110 PRINT "#2 TO LOAD THE PREVIOUS ENTRIES."
4130 PRINT :
     INPUT "PRESS ENTER";Z$:
     CLS :
     PRINT :
     PRINT :
     PRINT
4140 PRINT "3) UNDER TYPES OF INCOME, THE FIRST FOUR ARE TAXABLE, THE
      LAST"
4150 PRINT "FOUR NONTAXABLE AND ARE SO RECORDED BY THE PROGRAM."
4160 PRINT "MISCELLANEOUS TYPES SHOULD INCLUDE SUCH THINGS AS LOAN"
4170 PRINT "REPAYMENTS, GIFTS, REFUNDS, ETC.  MISCELLANEOUS ITEMS ARE
      NOT"
4180 PRINT "TOTALED INTO NONTAXABLE INCOME SINCE THEY DON'T REPRESENT
     "
```

```
4190 PRINT "INCOME.":
     PRINT
4200 PRINT "4) MENU CHOICES 4, 5 AND 6 MAY BE USED DURING PREPARATION
      OF"
4210 PRINT "MONTHLY FILES."
4220 PRINT :
     INPUT "PRESS ENTER";Z$:
     CLS
4230 PRINT :
     PRINT "5)  WHEN A MONTHLY FILE IS COMPLETE, USE DISK OPTION"
4240 PRINT "#1 TO SAVE IT TO DISK.  NEXT, GET YOUR PRINTER READY.  IF
      THIS"
4245 PRINT "RESULTS IN LOSS OF THE PROGRAM AND/OR MONTHLY FILE, RELOA
     D"
4250 PRINT "THEM.  THEN USE DISK OPTION #4 TO LOAD THE YEARLY TOTAL F
     ILE"
4255 PRINT "AND MENU OPTION #2 FOR COMPUTATIONS.  AT THIS TIME, THE R
     ESULTS"
4260 PRINT "MAY BE CHECKED ON VIDEO (MENU OPTION #6) OR HARD COPY MAD
     E"
4270 PRINT "(OPTION #7).  THE YEARLY TOTAL SHOULD BE SAVED TO DISK BE
     FORE"
4280 PRINT "LEAVING THE PROGRAM."
4290 PRINT
4310 PRINT "6) IT IS DESIRABLE TO KEEP A SET OF FILES ON A SECOND DIS
     K"
4320 PRINT "FOR BACKUP."
4330 PRINT :
     PRINT "7) REENTRY TO THE PROGRAM, IF NEEDED, IS AT LINE 100."
4340 PRINT :
     INPUT "PRESS ENTER";Z$:
     CLS :
     GOTO 40
6000 END
6099 REM  * CLASSIFY INCOME AND SEPARATE INTO TAXABLE, L(X),AND NONTA
     XABLE, R(X) *
6100 IF E(X) < > 0 PRINT E(X);
6110 PRINT :
     INPUT "AMOUNT OF EARNED INCOME";E(X)
6120 L(X) = E(X):
     GOTO 680
6200 IF D(X) < > 0 PRINT D(X);
6210 PRINT :
     INPUT "AMOUNT OF DIVIDEND";D(X)
6220 L(X) = D(X):
     GOTO 680
6300 IF I(X) < > 0 PRINT I(X);
6310 PRINT :
     INPUT "AMOUNT OF INTEREST";I(X)
6320 L(X) = I(X):
     GOTO 680
6400 IF A(X) < > 0 PRINT A(X);
6410 PRINT :
     INPUT "AMOUNT OF ANNUITY";A(X)
6420 L(X) = A(X):
     GOTO 680
6500 IF T(X) < > 0 PRINT T(X);
6510 PRINT :
     INPUT "AMOUNT OF TAX FREE INTEREST";T(X)
6520 R(X) = T(X):
     GOTO 680
6600 IF G(X) < > 0 PRINT G(X);
6610 PRINT :
     INPUT "AMOUNT OF SOCIAL SECURITY";G(X)
6620 R(X) = G(X):
     GOTO 680
6700 IF F(X) < > 0 PRINT F(X);
6710 PRINT :
     INPUT "AMOUNT OF LIFE INSURENCE DIVIDEND";F(X)
6720 R(X) = F(X):
```

Program continued

```
     GOTO 680
6800 IF C(X) < > 0 PRINT C(X);
6810 PRINT :
     INPUT "AMOUNT OF MISCELLANEOUS ITEM";C(X)
6820 R(X) = C(X):
     GOTO 680
7000 REM  * COMPUTATIONS *
7005 FOR X = 1 TO P1 - 1
7010  ET = ET + E(X):
      DT = DT + D(X):
      IT = IT + I(X):
      AT = AT + A(X)
7020  TT = TT + T(X):
      GT = GT + G(X):
      FT = FT + F(X)
7025  NEXT
7030 ES = ES + ET:
     DS = DS + DT:
     IS = IS + IT:
     AS = AS + AT
7040 TS = TS + TT:
     GS = GS + GT:
     FS = FS + FT
7050 LS = ES + DS + IS + AS
7060 RS = TS + GS + FS
7070 Q = LS + RS
7075 RETURN
```

Computacar

by R. A. Kay

When I began experimenting with I/O ports on my TRS-80 Level II, I soon became bored with flashing LEDs. The next thing to catch my fancy was an innocent radio-controlled racing car. What a great idea, I thought, to have a car hurtling around the basement under computer-radio-control.

This turned out to be simple to do. What's more, the car may still be used in its normal way.

Car Control

The car I use is one of several using a two-channel controller, providing left and right steering, but no stop/start/reverse. When I first hit RUN, I started running to catch the car before it demolished itself on the wall. It did not stop!

The inexpensive brand of car goes forward or does an abrupt turn while reversing. More expensive cars use extra channels to provide motion control such as forward/stop/reverse, or even proportional steering or speed. The following techniques can be expanded and will handle these features.

You don't have to understand how the radio link works, only how to enable the computer to simulate the action of the controller.

Figure 1 shows the switching arrangement to use the two channels. Switch A must always be closed for operation.

In the central position, switches B and C are open. Turning the wheel right closes switches B(1) and B(2), thus applying power (+ 9 volts) to the transmitter and the right modulator. This produces an output signal which is encoded by the car's receiver and causes the car to steer right. Turning the wheel left closes C(1) and C(2) and causes the car to steer left.

The steering mechanism is interesting. The front wheels normally run free, since drive is provided by the rear wheels. Upon receipt of a left or right control signal, the receiver actuates the appropriate wheel brake, which consists of a shaft rotated by the wheel within an electromagnet coil. A current in the coil causes the shaft to slow down, by attraction or eddy current. Thus, the wheel drags and pivots the front bogie to turn the car. A spring returns the bogie to the central position when the signal ceases.

Computer Control

To make the computer control the car requires only a two-bit output port (one bit per channel). When either bit is 1, a transistor applies + 9 volts to

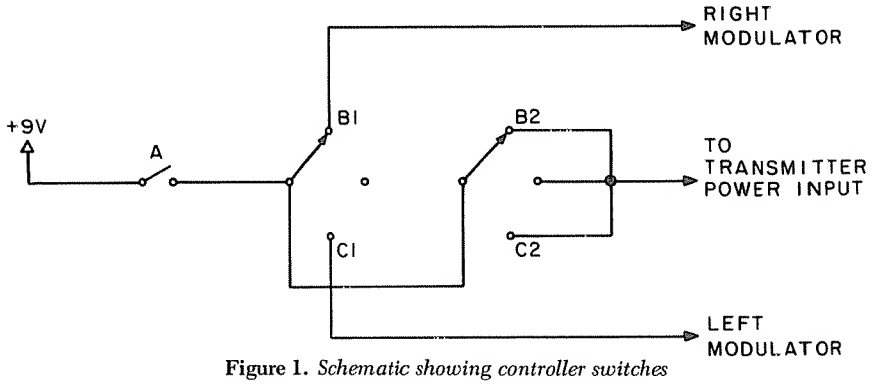


Figure 1. *Schematic showing controller switches*

D_0	D_1	ACTION
0	0	CENTRE
0	1	LEFT TURN
1	0	RIGHT TURN

Table 1. *Truth table of controller action in response to data outputs*

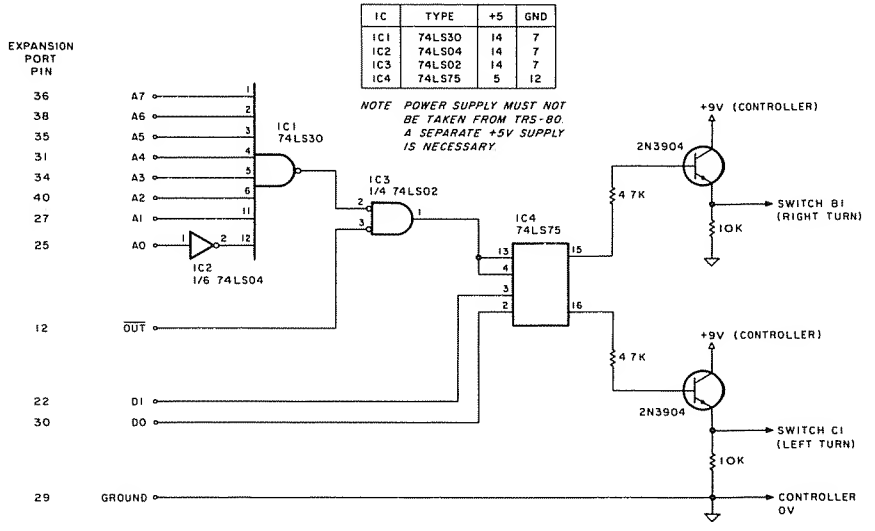


Figure 2. *Schematic of output port 254*

the appropriate switch contacts in the controller. A 0 bit opens the switch. Thus, we have a three-state controller as shown in the truth table, Table 1.

The TRS-80 allows 256 ports, numbered 0 to 255. Port 255 (FFH) is used by the cassette player. I chose to use port 254, since it requires the fewest gates for decoding. The circuit of the output port and transistor drivers is shown in Figure 2.

When the correct port address is decoded with the out signal going low, the output from IC3 latches data bits D_0 and D_1 to the output of IC4. A 1 at the base of transistor Q1 or Q2 turns the output on, applying + 9 V to the appropriate switch contacts in the controller. Switch A is simply shorted for continuous action.

Simple Program

Only 00, 01, and 10 are acceptable values for D_1D_0, corresponding to BASIC statements, in Level II:

```
OUT 254, 0   - Center
OUT 254, 1   - Left
OUT 254, 2   - Right
```

D_0 and D_1, both 1 (3 decimal), cannot be allowed, since these would cause the car to steer both ways. The car would probably end up going straight, with a certain amount of smoke!

The Program Listing shows a simple program to control the car on a predefined course, given as a string of steering commands (up to 250) in line 100. The program reads and holds each command for a period defined by the loop in line 160. The subroutine in lines 800-880 draws a plan of the course on the screen (within the limits of TRS-80 graphics). I find that five successive turn commands produce a 90 degree turn, hence the increment of turn (INC, line 800) is 90/5 = 18 degrees, converted to radians for use by the SIN and COS functions.

Here are some points to note:

- The power supply for the TRS-80 is barely adequate for the computer. Do *not* use it to power external circuits, such as I/O ports. A simple 5 V power supply is shown in Figure 3, which is adequate for the circuit here.
- Model cars are susceptible to dust, so racing surfaces must be clean and smooth. A tile floor is ideal. Any bumps or obstacles may affect the steering.
- Make sure that the course is a closed loop in range of the radio link (~30 foot radius). Otherwise the car will keep going once radio control is lost. Having antennae vertical ensures maximum range.

The car is "dumb" and tries to follow the course defined by the computer, but can't avoid obstacles. This would require it to have sensors, transmitting data back to the computer input ports.

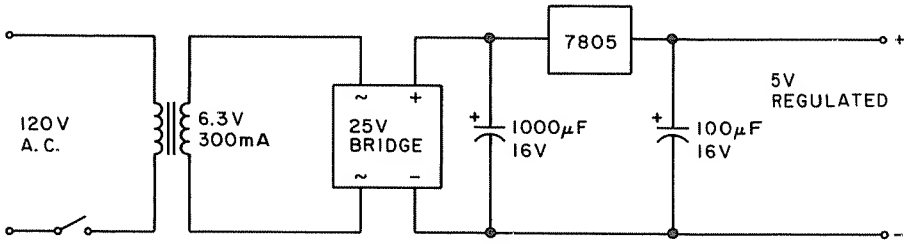


Figure 3. *5 V Regulated Power Supply*

This simple application shows the basics of operating a remote device via computerized radio control. And by using the same radio parts with different output devices, the possibilities are endless. For instance, how about a radio-controlled lawn-mower? All you need is a powered mower. Use the same radio control to operate steering devices. Input the appropriate course to the computer, type RUN, and watch your lawn being cut automatically.

Program Listing. *Simple Control Program*

```
10 REM  RADIO CAR CONTROL
20 REM
30 INSERT COMM AND STRING IN LINE 100 ( < 250 CHAR)
40 REM  0=STRAIGHT
50 REM  1=LEFT TURN
60 REM  2=RIGHT TURN
70 REM  >2=END
80 REM
90 REM  ADJUST DURATION OF COMMAND IN LINE 160
99 REM
100 A$ = "00000000111111000000002222222222220000000000000011111003"
110 N = 1:
    CLS
115 ANG = 0:
    X = 90:
    Y = 20
120 B$ = MID$(A$,N,1)
130 COM = VAL(B$)
140 IF COM > 2
     THEN
      999
150 OUT 254,COM
155 GOSUB 800
160 FOR M = 1 TO 100:
     NEXT M
170 N = N + 1
180 GOTO 120
799 END
800 INC = 18 * .0174533
810 IF COM = 1
     THEN
      ANG = ANG - INC
820 IF COM = 2
     THEN
      ANG = ANG + INC
830 X = X - 4 * COS(ANG)
840 Y = Y - SIN(ANG)
850 IF X < 0
     THEN
      X = 0
851 IF X > 127
     THEN
      X = 127
852 IF Y < 0
     THEN
      Y = 0
853 IF Y > 47
     THEN
      Y = 47
860 SET(X,Y)
870 SET(X - 1,Y)
880 RETURN
999 END
```

Bio-Bars: Biorhythms in Bar Graph Form

by Ronald J. Thibodeau

Here is a biorhythm program that does not need to be analyzed. If you are unhappy with the usual sine wave display currently being used for biorhythms, this bio-bars program may keep your interest up.

By now, almost everyone is familiar with biorhythms and what they mean. In theory, biorhythmic patterns describe our "ups and downs" in terms of our physical, emotional, and intellectual condition. Based on the research done by doctors Swoboda and Fliess, the biorhythm theory states that three cycles of 23, 28, and 33 days run concurrently from birth and continue until we die. The first half of each cycle represents an area of strength, while the second half of the cycle represents relative weakness. The physical cycle lasts for a period of 23 days. The first half (11½ days) is a high period while the second half is considered to be a low period of activity. Similarly, the emotional cycle lasts for 28 days (14 high and 14 low) and the intellectual cycle lasts for 33 days (16½ high and 16½ low). These cycles can be represented in the form of a bar graph as shown in Figure 1.

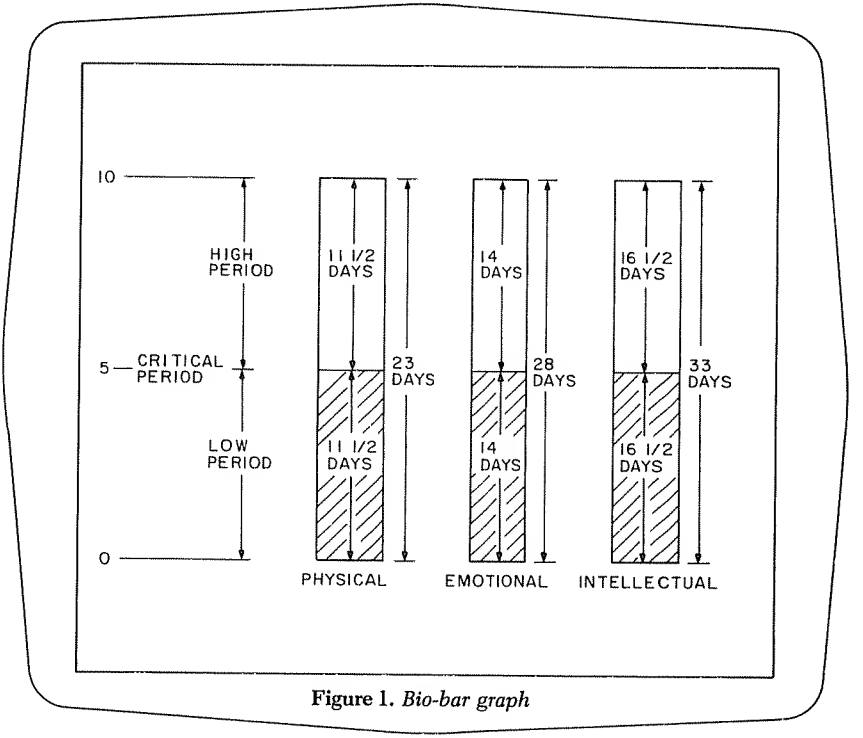


Figure 1. *Bio-bar graph*

In the bar graph presentation, the high period occurs above a value of five and up to and including a value of ten. The low period occurs between a value of zero and up to but not including a value of five. The midpoint of the scale, five, represents a critical period when the body is adjusting (or trying to adjust) to a transition from a high period of performance to a low period of performance and vice-versa.

If a bar is at level five it is considered to be a "critical" day. If two bars are at level five it is considered a double-critical period. When all three bars are at level five, it is a triple-critical day. These critical days are not considered dangerous but they do indicate that a person's reactions may bring about an unsatisfactory situation.

Since the direction of the bar could be increasing (going up) or decreasing (going down), we must have a way of denoting this direction. To do this, the program produces an up or down arrow to the left of each bar. See Figure 2.

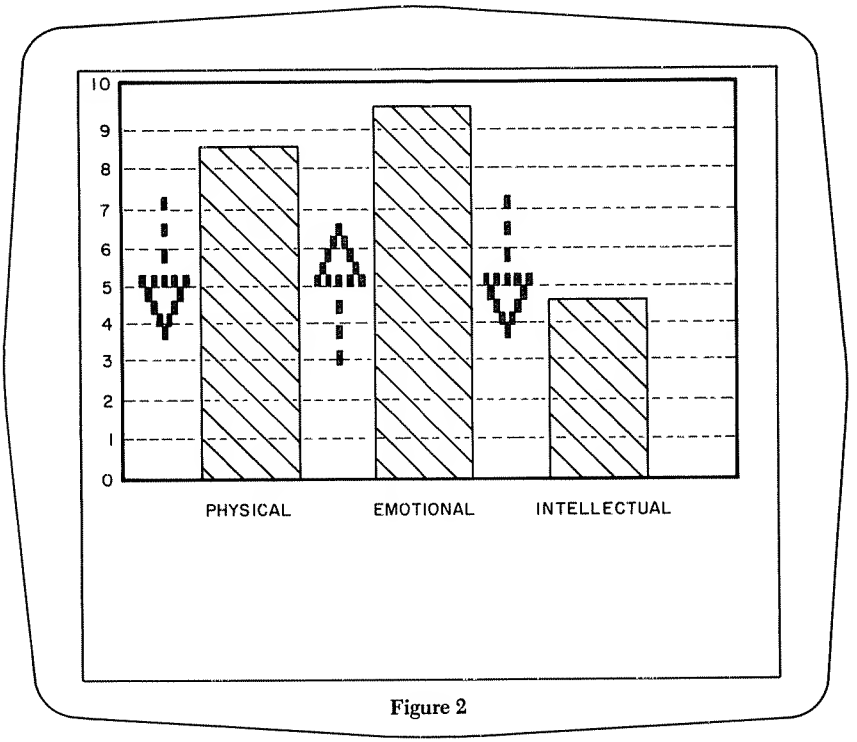


Figure 2

This program was originally written for the TRS-80 Level I, but was modified for Level II as explained in Appendix A. If a 4K system is used, you will have to be very careful not to use excess spaces since the program will require all but 69 bytes.

Lines 10-40

These are simply an introduction to the program and may be eliminated if you need more memory.

Lines 100-130

The only inputs required in the program appear in these lines. They are: (1) name, (2) date of birth, (3) date of computation. You can enter any date as a date of computation. But, if you enter a date in the future, don't expect the computer to be your personal crystal ball. Biorhythm programs do not predict what will happen, but rather how you might behave. If the date of birth and the date of computation occur during the same century, you can enter the year by using only the last two significant digits. However, if the date of birth and the date of computation are in different centuries, you must enter the entire year (1980 instead of 80).

Lines 140-145

Both lines are simple FOR-NEXT loops to produce the scale numbers from one through ten and the dashed lines across the screen. See Figure 3.

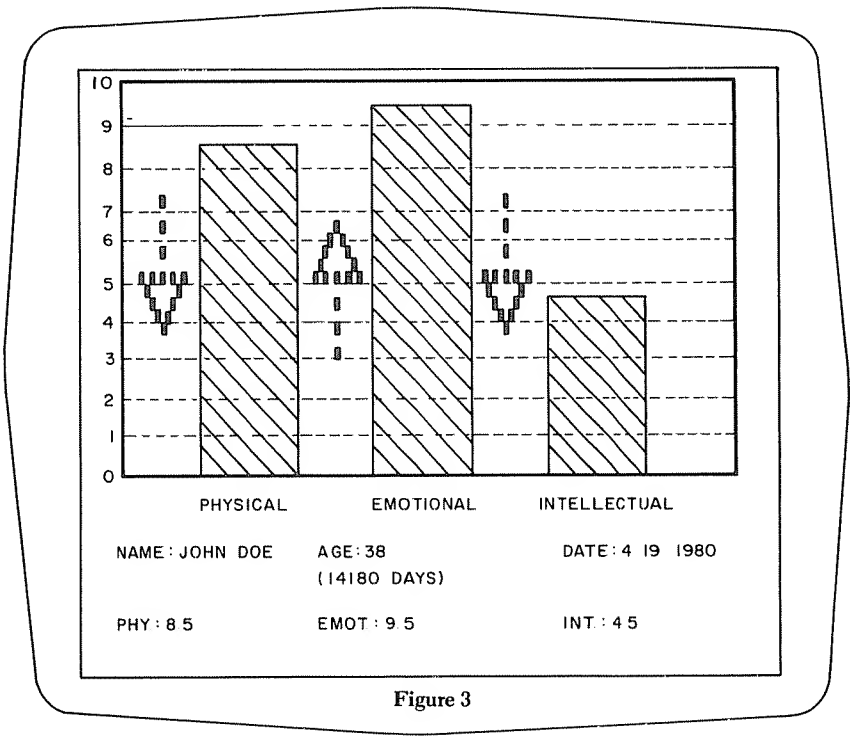


Figure 3

Lines 150-170

These lines produce the bar graph border and the titles of each bar (PHYSICAL, EMOTIONAL, INTELLECTUAL). Be sure to include eight spaces between physical and emotional and six spaces between the emotional and intellectual titles for proper alignment with each bar. See Figure 3.

Lines 180-230

This portion computes the total number of days that input A (person) has lived. It begins by computing the person's age, lines 180-190, and adding the number of days elapsed since the birthday to the day of computation (lines 215-225). Leap year add-on days are added in line 230 by the variable INT G/4.

Lines 240-270

These lines divide the total number of days by the cycles of 23, 28, and 33. The remainder of this quotient is then used in the next portion of the program. Line 270 contains constants which must not be altered.

Lines 280-400

All of the IF-THEN statements included here will convert the variables I, J, and K into corresponding values between zero and ten (L,M,N). These are later converted into the appropriate bar heights for each cycle.

Lines 500-550

The variable P is used to convert the values of L, M, and N into bar heights using the video graphics Y-coordinate. The X-coordinate is simultaneously assigned to locate the bars across the screen.

Lines 560-650

Line 560 produces the base for each of the three arrows. Lines 600 through 650 determine the direction (up or down) for each arrow. The GOSUB-1000 routine produces the remainder of each arrow based on the value of variable T. Be very careful not to confuse the numeral zero with the variable letter O in this portion of the program.

Lines 660-670

After the video graphics are produced, these lines will print all of the inputs entered earlier along with the person's age, in years and days, and the date of computation. The actual values of the bar heights, values of zero to ten, are also shown because the bar heights are only an approximation. See Figure 3.

Lines 735-980 and 2000-2130

These lines provide statements regarding the person's physical, emotional, and intellectual condition. A calculation is also made and displayed to show the duration of days remaining in a cycle. All critical days (value of five) last for 24 hours and no computation is required. See Figure 4.

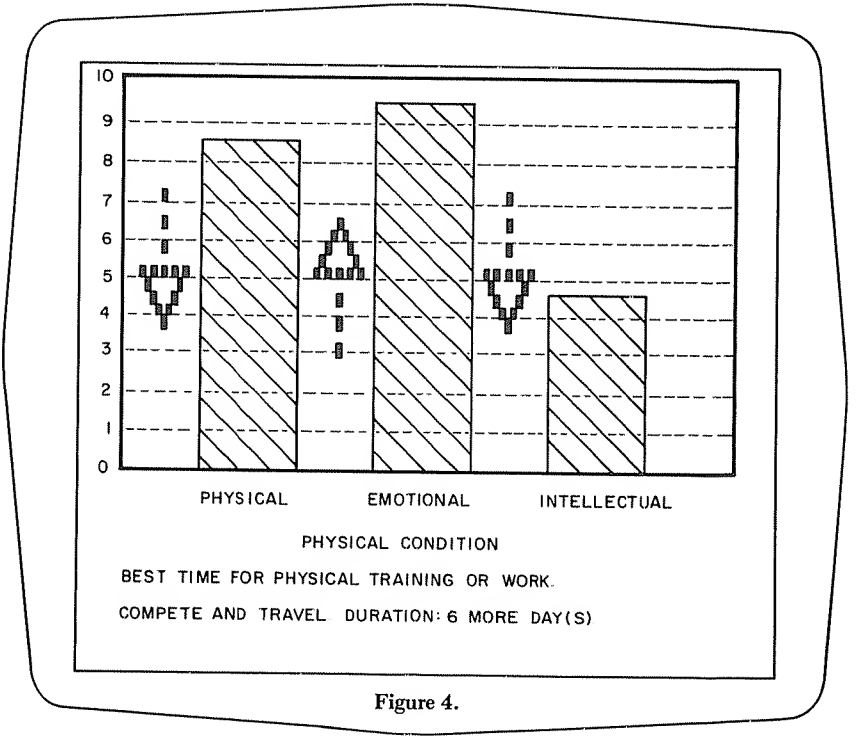


Figure 4.

Subroutines 1000 and 3000

Subroutine 1000 draws the up and down arrows. If the value of T is unspecified, then lines 1010-1030 will be executed to produce the up arrow. If T = 1, then lines 1040-1060 will execute the down arrow.

Subroutine 3000 provides a time delay to allow the reading of each biorhythm statement. It then erases the screen from locations 768 to 1023 so that the next statement will appear in legible condition.

At the end of a program, lines 2050, 2095, and 2140 send command back to the beginning of the program so that someone else can obtain a graph.

Good luck and may all your biorhythms be favorable.

Program Listing

This listing was originally in Level I.

```
 1 DIM A(12)
 2 REM  DIM UNNECESSARY FOR LEVEL I
10 CLS :
   A = 2:
   B = 54:
   FOR N = 1 TO 8:
    PRINT TAB(A),"BIO-BARS", TAB(B),"BIO-BARS"
20  A = A + 3:
    B = B - 3:
    NEXT N:
   PRINT TAB(28),"BIO-BARS":
   PRINT TAB(30),"BIO":
   PRINT TAB(31),"B"
30 PRINT :
   PRINT TAB(24),"BIO-BARS PROGRAM":
   PRINT TAB(28),"BY TIBBY"
40 FOR N = 1 TO 2000:
    NEXT N
100 CLS
110 INPUT "ENTER YOUR NAME ";A$
120 INPUT "ENTER DATE OF BIRTH (EX.6,14,1941).USE COMMAS.";A,B,C
130 INPUT "ENTER TODAY'S DATE AS ABOVE.";D,E,F
140 CLS :
    U = - 1:
    V = 704:
    W = 64:
    FOR N = 1 TO 11:
     U = U + 1:
     V = V - 64:
     PRINT @V,U:
     NEXT N
145 H = 69:
    I = 123:
    FOR N = 1 TO 9:
     FOR X = H TO I:
      PRINT @X,"-":
      NEXT X:
     H = H + 64:
     I = I + 64:
     NEXT N
150 FOR X = 7 TO 8:
     FOR Y = 0 TO 31:
      SET(X,Y):
      SET(X + 115,Y):
      NEXT Y:
     NEXT X
160 FOR X = 7 TO 122:
     SET(X,0):
     SET(X,31):
     NEXT X
170 PRINT @715,"PHYSICAL--------EMOTIONAL------INTELLECTUAL"
180 IF (D > A) + ((D = A) * (E > = B))
     THEN
      G = F - C
190 IF (D < A) + ((D = A) * (E < B))
     THEN
      G = F - C - 1
200 A(1) = 31:
    A(2) = 59:
    A(3) = 90:
    A(4) = 120:
    A(5) = 151:
    A(6) = 181:
    A(7) = 212
210 A(8) = 243:
```

Program continued

```
    A(9) = 273:
    A(10) = 304:
    A(11) = 334:
    A(12) = 365
215 IF (A < D) + ((A = D) * (E > B))
     THEN
      Z = A(D) - A(A) + E - B
220 IF (A > D) + ((A = D) * (E < B))
     THEN
      Z = 365 - (A(A) - A(D)) - B + E
225 IF (A = D) * (E = B)
     THEN
      Z = 0
230 H = (G * 365) + ( INT(G / 4)) + Z
240 I = INT(((H / 23) - INT(H / 23)) * 23)
250 J = INT(((H / 28) - INT(H / 28)) * 28)
260 K = INT(((H / 33) - INT(H / 33)) * 33)
270 U = .8695:
    V = .7142:
    W = .606:
    IF I = 23
     THEN
      I = 0
280 IF (I > = 0) * (I < = 5.75)
     THEN
      L = (U * I) + 5
290 IF (I > 5.75) * (I < = 17.25)
     THEN
      L = ( - U * I) + 15
300 IF (I > 17.25) * (I < 23)
     THEN
      L = (U * I) - 15
320 IF J = 28
     THEN
      J = 0
330 IF (J > = 0) * (J < = 7)
     THEN
      M = (V * J) + 5
340 IF (J > 7) * (J < = 21)
     THEN
      M = ( - V * J) + 15
350 IF (J > 21) * (J < 28)
     THEN
      M = (V * J) - 15
370 IF K = 33
     THEN
      K = 0
380 IF (K > = 0) * (K < = 8.25)
     THEN
      N = (W * K) + 5
390 IF (K > 8.25) * (K < = 24.75)
     THEN
      N = ( - W * K) + 15
400 IF (K > 24.75) * (K < 33)
     THEN
      N = (W * K) - 15
500 P = INT(31 - L * 3)
510 FOR X = 22 TO 39:
     FOR Y = 31 TO P STEP - 1:
      SET(X,Y):
      NEXT Y:
     NEXT X
520 P = INT(31 - M * 3)
530 FOR X = 54 TO 71:
     FOR Y = 31 TO P STEP - 1:
      SET(X,Y):
      NEXT Y:
     NEXT X
540 P = INT(31 - N * 3)
```

```
550 FOR X = 86 TO 103:
     FOR Y = 31 TO P STEP - 1:
      SET(X,Y):
      NEXT Y:
     NEXT X
560 FOR X = 11 TO 19 STEP 2:
     Y = 15:
     SET(X,Y):
     SET(X + 32,Y):
     SET(X + 64,Y):
     NEXT X
600 T = 0:
    O = 11:
    IF (I > 5.75) * (I < 17.25)
     THEN
      T = 1
610 GOSUB 1000
620 T = 0:
    O = 43:
    IF (J > 7) * (J < 21)
     THEN
      T = 1
630 GOSUB 1000
640 T = 0:
    O = 75:
    IF (K > 8.25) * (K < 24.75)
     THEN
      T = 1
650 GOSUB 1000
660 PRINT "NAME :";A$; TAB(22),"AGE :";G; TAB(44),"DATE :";D;E;F
665 PRINT TAB(22),"(";H;"DAYS)"
670 PRINT "PHY. :";L; TAB(22),"EMOT. :";M; TAB(44),"INT. :";N;
735 GOSUB 3000
740 PRINT @790,"PHYSICAL CONDITION"
750 IF L < = 5 GOTO 790
760 PRINT "BEST TIME FOR PHYSICAL TRAINING OR WORK.
770 PRINT "COMPETE AND TRAVEL. DURATION :";11.5 - I;"MORE DAY(S).
780 GOSUB 3000
785 GOTO 860
790 IF L = 5 GOTO 830
800 PRINT "REST AND RECHARGE YOUR BATTERY. TENDENCY TO TIRE EASILY.
810 PRINT "GENERAL LACK OF ENERGY. DURATION :";23 - I;"MORE DAY(S).
820 GOSUB 3000
825 GOTO 860
830 PRINT "CRITICAL ! UNSTABLE. TREAT FOR HEADACHES, ILLNESSES,ETC.
840 PRINT "ACCIDENT PRONE. DURATION : 24 HOURS.
850 GOSUB 3000
860 PRINT @790,"EMOTIONAL CONDITION"
870 IF M < = 5 GOTO 910
880 PRINT "OPTIMISTIC, GOOD NATURED WITH HIGH CREATIVITY. GOOD FOR
890 PRINT "TEAMWORK AND SEX. DURATION :";14 - J;"MORE DAY(S)."
900 GOSUB 3000
905 GOTO 2000
910 IF M = 5 GOTO 950
920 PRINT "TENDENCY TO BE IRRITABLE AND UNCOOPERATIVE.
930 PRINT "RECHARGE. DURATION :";28 - J;"MORE DAY(S).
940 GOSUB 3000
945 GOTO 2000
950 PRINT "CRITICAL ! UNSTABLE EMOTIONS. LOW REACTIONS. QUARRELS.
960 PRINT "KEEP COOL. DURATION : 24 HOURS.
970 GOSUB 3000
980 GOTO 2000
1000 IF T = 1 GOTO 1040
1010 X = O:
     Y = 15:
     FOR Q = 1 TO 5:
      SET(X,Y):
      Y = Y - 1:
      X = X + 1:
```

Program continued

```
      NEXT Q
1020 X = O + 8:
     Y = 15:
     FOR Q = 1 TO 5:
      SET(X,Y):
      Y = Y - 1:
      X = X - 1:
      NEXT Q
1030 X = O + 4:
     FOR Y = 15 TO 19 STEP 2:
      SET(X,Y):
      NEXT Y
1035 RETURN
1040 X = O:
     Y = 15:
     FOR Q = 1 TO 5:
      SET(X,Y):
      Y = Y + 1:
      X = X + 1:
      NEXT Q
1050 X = O + 8:
     Y = 15:
     FOR Q = 1 TO 5:
      SET(X,Y):
      Y = Y + 1:
      X = X - 1:
      NEXT Q
1060 X = O + 4:
     FOR Y = 15 TO 11 STEP - 2:
      SET(X,Y):
      NEXT Y
1070 RETURN
2000 PRINT @788,"INTELLECTUAL CONDITION"
2010 IF N < = 5 GOTO 2060
2020 PRINT "EXCELLENT THINKING, JUDGEMENT AND CONCENTRATION.
2030 PRINT "PLAN AND DECIDE. DURATION :";16.5 - K;"MORE DAY(S).
2040 GOSUB 3000
2050 GOTO 10
2060 IF N = 5 GOTO 2100
2070 PRINT "LACKING IN THINKING POWER. GATHER AND ADJUST.
2080 PRINT "DO EASY TASKS. DURATION :";33 - K;"MORE DAY(S).
2090 GOSUB 3000
2095 GOTO 10
2100 PRINT "CRITICAL ! UNSTABLE INTELLECTUAL POWERS. MEMORY FAILS.
2120 PRINT "LOW MENTAL ABILITIES. DURATION : 24 HOURS.
2130 GOSUB 3000
2140 GOTO 10
3000 FOR X = 1 TO 5000:
      NEXT X:
     FOR X = 768 TO 1023:
      PRINT @X,:
      NEXT X
3010 RETURN
9999 END
```

INTERFACE

TTY Interface

Why Bother to Interface?

TTY Interface

by Robin Rumbolt

After a few weeks of familiarization with my TRS-80 and Radio Shack's Level II BASIC, I found my programs getting longer than the 16 lines that the CRT would accommodate at one time. Listing and relisting to see various parts of the program during debugging became tedious. I soon found myself making repeated trips to the neighborhood Radio Shack store to get my programs listed on their line printer.

This couldn't go on. After all, I did have that model 33 Teletype out in the garage if I could figure out how to hook it up. I found several Teletype interface kits on the market in the $75-$120 price range, but having barely squeaked the TRS-80 into the family budget and being an avid homebrewer, I decided to build my own interface. The resulting circuit required only five parts.

Interfacing Techniques

The usual method of interfacing a Teletype to the TRS-80 is shown in Figure 1. This method requires a UART that accepts data in parallel form from the CPU data bus, adds the proper start and stop bits, and sends the data out in serial form at a rate determined by an external baud rate generator. This method is fast and efficient and, if interrupt driven, can even let the processor do something else while the UART is serializing the data. For parts, it requires an address decoder circuit, a UART, bus buffers, a baud rate generator IC, and some form of TTL-to-current-loop interface circuit. Also, a software routine is required to set up the internal functions of the UART and then continuously route data to the UART.

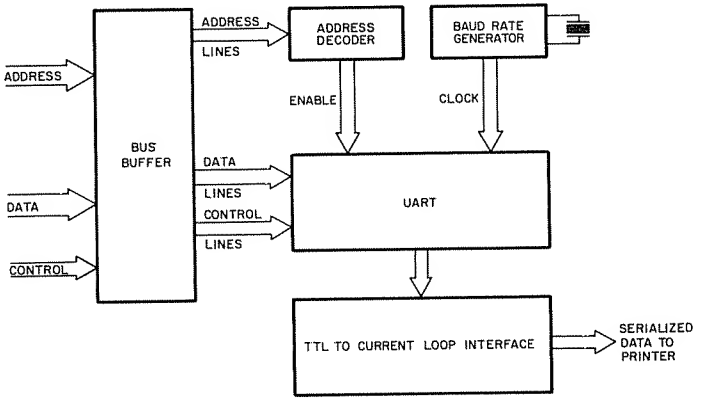


Figure 1. *Hardware printer interface block diagram*

Another method, shown in Figure 2, merely writes a bit at a time out to a location at a decoded I/O port. This method requires hardware only for decoding the port address and interfacing to the current loop of the printer. While this method requires less hardware than the first method, it does require more software. In addition, the processor is tied up the whole time doing the actual output of bits. The software routine must take the data, add the proper start and stop bits, and then write the bits out to the I/O port one at a time at a rate determined by a software timing loop.

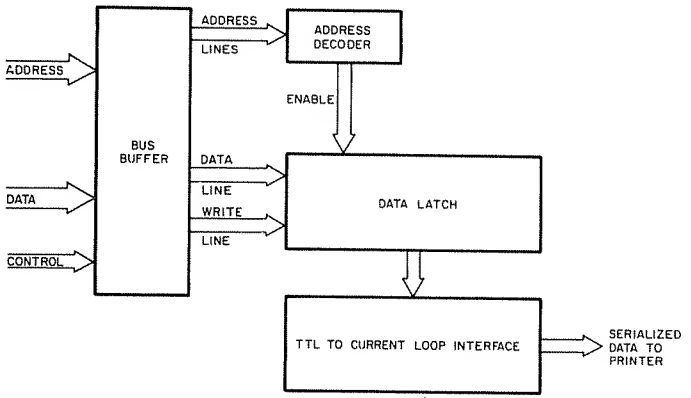


Figure 2. *Alternate interface block diagram*

Since the LPRINT routine in the TRS-80 ROM is non-interrupt driven, and since I had enough half-finished construction projects in process already, it was obvious that the simplest hardware interface was the best.

Remembering that the CPU decodes port FF for cassette input/output and motor control, I studied the schematic for any unused bits I might use in that port. There were several. I decided to use bit D5 for my serial output line. Since the port was already decoded, the need for external address decoding and bus-buffering hardware was eliminated. Now all I had to provide was a data latch to store the bit and a current loop interface.

Figure 3 shows how I accomplished this task with only five parts. The 74LS175, which I will refer to as Z59A, serves as the data latch. Although it is capable of latching four bits, I used only one section to latch bit D5. Resistors R1 and R2 and transistor Q1 are used to buffer the output of the latch and provide approximately 20 mA of drive current for the optical isolator.

The 4N35 optical isolator provides about 2500 volt isolation between the outside world current loop and my precious CPU—a nice thing to have when you accidentally connect the loop supply wrong! The outputs of the 4N35 are connected to two unused pins on the video output connector. These pins then connect to the 20 mA printer loop.

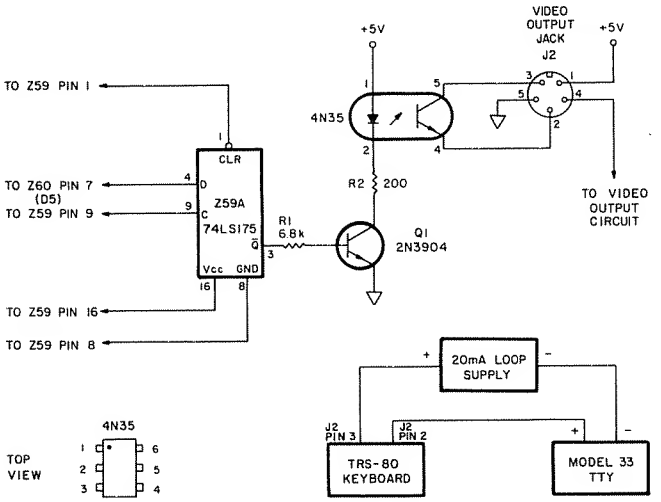


Figure 3. *TTY interface schematic*

Software

Now that I had the hardware capability to enable or disable current flow in an external loop, I needed a program to turn this current flow on and off according to the ASCII bit pattern required by the printer. The flowchart of such a program is shown in Figure 4.

Since the normal logic state of the bits in port FF is 0, I used the logic 0 to represent the mark, or current-flowing condition, in the loop. A logical 1 written to the output port represents the space condition and disables current flow in the loop. The flowchart describes the program fairly well, so I will only touch on a few of the high points.

The LPRINT and LLIST routines in ROM never output a line-feed code since they assume the printer to have automatic line feed on the carriage return. Since my machine does not have that feature, I included a routine to output a line feed whenever a carriage return was output by the ROM.

Also included is a character-counter routine set to limit line length to approximately 70 characters. I use memory location 16424 as the counter. Every time a character is output, this memory location is incremented. When it reaches 70, a carriage return and line feed are printed. The character counter is set to zero initially and also every time a carriage return/line feed is output.

Because the LPRINT and LLIST routines were written to operate with Centronics printers, they examine the line printer address, memory location 14312, for a ready bit to be set before they output the next character.

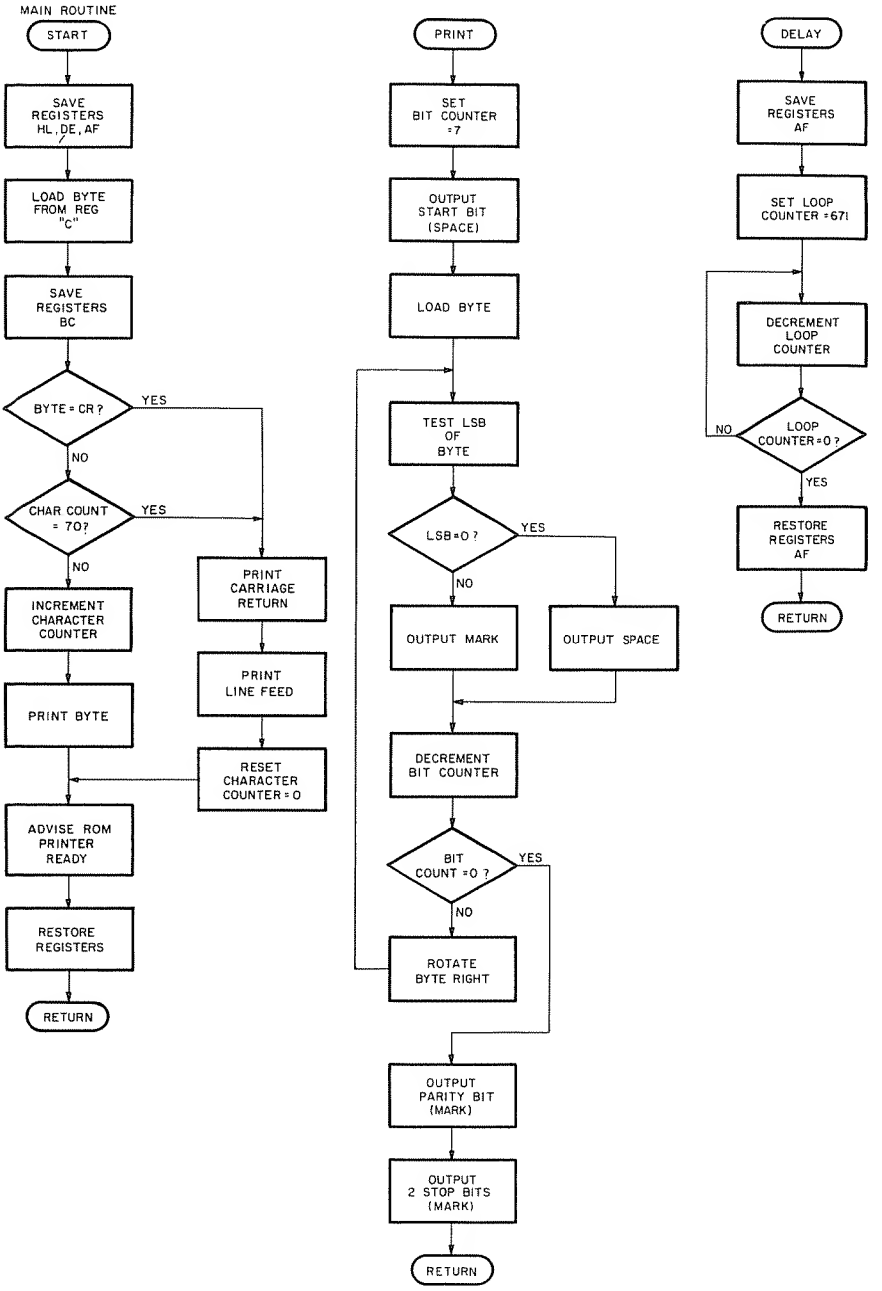


Figure 4. *Software flowchart*

Therefore, the Teletype program writes a fake ready bit to this address to enable a character output by the ROM routines.

The section of code labeled PRINT in Figure 4 is a part of the program that serializes the data to be output. It first holds the output in space condition for one bit time to simulate the start bit. Then the program samples each bit in register H and outputs a logic 0 (mark) or a logic 1 (space) to bit D5 of port FF, depending on the bit status of the byte being printed. After all bits have been sampled, the program holds the output in mark condition for three bit times—one for an even parity bit and two for two stop bits—before executing a return to ROM.

The part of the program labeled DELAY determines the baud rate of the output. The delay loop furnishes approximately 13.56 microseconds' delay each time it is executed. Therefore, for the 9.1 millisecond bit time required for a 110 baud printer, this delay loop is executed 671 times.

Installation

To make the required modifications, first place the keyboard face down on a soft surface and remove the six Phillips-head screws holding the case together. Note that there are three different sizes, so note where each size goes. Next, holding the case together with your hands, place the keyboard right side up on your work surface and gently lift off the top cover.

Notice that the keyboard is connected to the main PC board by a delicate, white, flat ribbon cable located on the lower left edge of the keyboard PC board. Trying not to flex this cable too much, pull the keyboard away from the main PC board and hold it vertically while removing the five little white plastic spacers that are on the PC board and lift the entire assembly out of the bottom half of the case. Holding the keyboard against the main PC board, turn the whole assembly over and place it so the IC numbers can be read with the keyboard face down.

Locate Z59 in the lower left area. Take a new 74LS175 IC and bend all but the four corner pins outward. Place this IC, Z59A, piggyback style on top of Z59. Solder the four corner pins of Z59A to their counterparts on Z59. Use as little heat and solder as possible to avoid overheating the ICs or causing shorts.

Now cut one of the leads of both R1 and R2 and the base and collector leads of Q1 to a length of about ¼ inch. Solder the short lead of R1 to the base lead of Q1. Similarly, solder the short lead of R2 to the collector of Q1. Turn Q1 upside down and solder its emitter lead to Z59, pin 8. Run a short piece of insulated wire from Z59A, pin 4, to Z60, pin 7. Make sure that all unconnected leads of Z59A are bent away from Z59 and are not touching anything else. Solder one end of a long piece of insulated wire to the unconnected lead of R2. Take a good look at what you have done so far. Make sure

that the transistor and resistors are secure and that their leads are not touching anything that they are not supposed to touch.

Once satisfied, turn the PC boards over and mount them back in the lower half of the case. Before putting the top cover back on, do the following: Take a 4N35 IC and cut off pins 3 and 6. Then, referring to Figure 5, solder pin 1 of the 4N35 to the trace shown. Solder the end of the hookup wire still dangling to pin 2 of the 4N35. Solder pins 4 and 5 of the 4N35 to short pieces of hookup wire, which should then be soldered to the video connector pins shown.

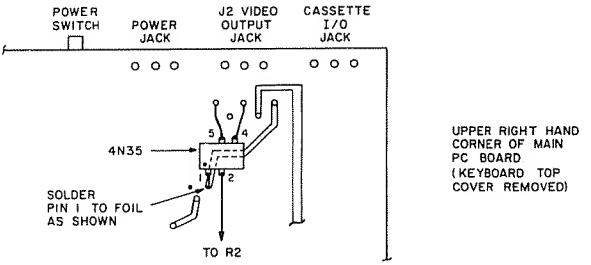


Figure 5. *4N35 installation on main PC board*

The modification is now complete. Reassemble the rest of the case. Connect the Teletype and loop supply as shown. Upon power-up, set MEMORY SIZE to 32566 for a 16K machine or 20277 for a 4K model. Enter and run the TTY program. When RUN is typed, the BASIC program will POKE the TTY program into high memory and then destroy itself. From then on, the LLIST and LPRINT statements should work using the Teletype as a line printer.

Conclusion

For those who are not afraid to modify their prized equipment, this modification is convenient and works well with any program utilizing the line printer control block set up in RAM by the Level II ROM. Since Radio Shack's Editor/Assembler does not use this control block, the printer will not operate while using the Editor/Assembler. This is the only shortcoming I have found.

A final note: As I was going over the program for the 50th time in preparation for this article, I noticed that by adding four more bytes to the beginning of the program I could have the CRT echo what was being LPRINTed. I have included them in the TTY programs. These bytes are a call to a subroutine located in ROM at hex location 0033, which displays the character on the CRT.

References

- *Radio Shack TRS-80 Editor/Assembler Operation and Reference Manual.* Ft. Worth, TX: Radio Shack, 1978.
- *TRS-80 Technical Manual.* Ft. Worth, TX: Radio Shack, 1978.

Program Listing 1. *TTY program for 4K Level II TRS-80*

```
 5 :
   ' TTY 4K LEVEL II-----SET MEM SIZE TO 20277
10 CLS :
   INPUT "DO YOU WANT CRT TO ECHO LPRINT?";A$:
   IF A$ < > "Y" AND A$ < > "N"
    THEN
     10 :
    ELSE
     IF A$ = "Y"
      THEN
       POKE 16422,54 :
      ELSE
       POKE 16422,58
20 POKE 16423,79:
   POKE 16424,0:
   FOR I = 20278 TO 20423
30  READ BYTE
40  POKE I,BYTE
50  NEXT
60 DATA 121,205,51,0,229,213,245,97,197,124,87,254,13,194,86,79,205
   ,136,79,38
65 DATA 10,205,136,79,62,0,50,40,64,195,126,79,58,40,64,254,70,194,
   115,79,205
70 DATA 136,79,38,13,205,136,79,38,10,205,136,79,62,0,50,40,64,195,
   126,79,58
75 DATA 40,64,60,50,40,64,98,205,136,79,62,63,50,232,55,193,241,209
   ,225,201,6
80 DATA 7,62,48,211,255,205,188,79,124,103,203,71,202,157,79,62,16,
   195,159,79
85 DATA 62,48,211,255,205,188,79,5,202,174,79,124,203,31,195,146,79
   ,62,16,211
90 DATA 255,205,188,79,205,188,79,205,188,79,201,245,17,159,2,27,12
   2,179,194
95 DATA 192,79,241,201
97 NEW
99 END
```

Program Listing 2. *TTY program for 16K Level II TRS-80*

```
 5 :
   ' TTY 16 K LEVEL II-----SET MEM SIZE TO 32566
10 CLS :
   INPUT "DO YOU WANT CRT TO ECHO LPRINT?";A$:
   IF A$ < > "Y" AND A$ < > "N"
    THEN
     10 :
    ELSE
     IF A$ = "Y"
      THEN
       POKE 16422,55 :
      ELSE
       POKE 16422,59
20 POKE 16423,127:
   POKE 16424,0:
   FOR I = 32567 TO 32712
30  READ BYTE
40  POKE I,BYTE
50  NEXT
```

```
60 DATA 121,205,51,0,229,213,245,97,197,124,87,254,13,194,87,127,20
   5,137,127,38
65 DATA 10,205,137,127,62,0,50,40,64,195,127,127,58,40,64,254,70,19
   4,116,127,205
70 DATA 137,127,38,13,205,137,127,38,10,205,137,127,62,0,50,40,64,1
   95,127,127,58
75 DATA 40,64,60,50,40,64,98,205,137,127,62,63,50,232,55,193,241,20
   9,225,201,6,7
80 DATA 62,48,211,255,205,189,127,124,103,203,71,202,158,127,62,16,
   195,160,127,62
85 DATA 48,211,255,205,189,127,5,202,175,127,124,203,31,195,147,127
   ,62,16,211
90 DATA 255,205,189,127,205,189,127,205,189,127,201,245,17,159,2,27
   ,122,179,194
95 DATA 193,127,241,201
97 NEW
99 END
```

Program Listing 3. *Assembly-language listing*

```
4F36            00001          ORG     4F36H
4F36 79         00002          LD      A,C
4F37 CD3300     00003          CALL    0033H
4F3A E5         00004          PUSH    HL
4F3B D5         00005          PUSH    DE
4F3C F5         00006          PUSH    AF
4F3D 61         00007          LD      H,C
4F3E C5         00008          PUSH    BC
4F3F 7C         00009          LD      A,H
4F40 57         00010          LD      D,A
4F41 FE0D       00011          CP      0DH
4F43 C2564F     00012          JP      NZ,EOL
4F46 CD884F     00013          CALL    PRINT
4F49 260A       00014          LD      H,0AH
4F4B CD884F     00016          CALL    PRINT
4F4E 3E00       00017          LD      A,00H
4F50 322840     00018          LD      (4028H),A
4F53 C37E4F     00019          JP      EXIT
4F56 3A2840     00020 EOL      LD      A,(4028H)
4F59 FE46       00021          CP      46H     ;="F"
4F5B C2734F     00022          JP      NZ,MIN
4F5E CD884F     00023          CALL    PRINT
4F61 260D       00024          LD      H,0DH
4F63 CD884F     00025          CALL    PRINT
4F66 260A       00026          LD      H,0AH
4F68 CD884F     00027          CALL    PRINT
4F6B 3E00       00028          LD      A,00H
4F6D 322840     00029          LD      (4028H),A
4F70 C37E4F     00030          JP      EXIT
4F73 3A2840     00031 MIN      LD      A,(4028H)
4F76 3C         00032          INC     A
4F77 322840     00033          LD      (4028H),A
4F7A 62         00034          LD      H,D
4F7B CD884F     00035 OUT      CALL    PRINT
4F7E 3E3F       00036 EXIT     LD      A,3FH   ;="?"
4F80 32E837     00037          LD      (37E8H),A
4F83 C1         00038          POP     BC
4F84 F1         00039          POP     AF
4F85 D1         00040          POP     DE
4F86 E1         00041          POP     HL
4F87 C9         00042          RET
4F88 0607       00043 PRINT    LD      B,07H
4F8A 3E20       00044          LD      A,20H   ;="0"
4F8C D3FF       00045          OUT     (0FFH),A
```

Program continued

```
4F8E CDBC4F    00046          CALL     DELAY
4F91 7C        00047          LD       A,H
4F92 67        00048 LOOP     LD       H,A
4F93 CB47      00049          BIT      0,A
4F95 CA9D4F    00050          JP       Z,SPACE
4F98 3E00      00051          LD       A,0
4F9A C39F4F    00052          JP       BIT
4F9D 3E20      00053 SPACE    LD       A,20H   ;=" "
4F9F D3FF      00054 BIT      OUT      (0FFH),A
4FA1 CDBC4F    00055          CALL     DELAY
4FA4 05        00056          DEC      B
4FA5 CAAE4F    00057          JP       Z,DONE
4FA8 7C        00058          LD       A,H
4FA9 CB1F      00059          RR       A
4FAB C3924F    00060          JP       LOOP
4FAE 3E00      00061 DONE     LD       A,0
4FB0 D3FF      00062          OUT      (0FFH),A
4FB2 CDBC4F    00063          CALL     DELAY
4FB5 CDBC4F    00064          CALL     DELAY
4FB8 CDBC4F    00065          CALL     DELAY
4FBB C9        00067          RET
4FBC F5        00068 DELAY    PUSH     AF
4FBD 119F02    00069          LD       DE,029FH
4FC0 1B        00070 DLOOP    DEC      DE
4FC1 7A        00071          LD       A,D
4FC2 B3        00072          OR       E
4FC3 C2C04F    00073          JP       NZ,DLOOP
4FC6 F1        00074          POP      AF
4FC7 C9        00075          RET
0000           00076          END
00000 TOTAL ERRORS
```

```
BIT     4F9F 00054    00052
DELAY   4FBC 00068    00046 00055 00063 00064 00065
DLOOP   4FC0 00070    00073
DONE    4FAE 00061    00057
EOL     4F56 00020    00012
EXIT    4F7E 00036    00019 00030
LOOP    4F92 00048    00060
MIN     4F73 00031    00022
OUT     4F7B 00035
PRINT   4F88 00043    00013 00016 00023 00025 00027 00035
SPACE   4F9D 00053    00050
```

```
PRINT :
PRINT "SET TOP OF FORM TO 20":
PRINT "SET LEFT EDGE OF FORM TO -4":
PRINT "SET DENSITY CONTROL TO 4":
INPUT " 'ENTER' TO CONTINUE";X
```

Why Bother to Interface?

by Allan S. Joffe W3KBM

Sooner or later you will want to control something in the way of an outside world device by using your TRS-80. It may be a relay to key a transmitter with Morse code or a method of turning lights on and off according to a predetermined schedule. Whatever your desire, the key to fulfillment is an interface. The other side of the coin is taking an outside world signal, properly conditioning it, and being able to feed it into the computer. The end result may be as fundamental as printing the input on the video screen or having the computer accept this information and then act on it by controlling a relay. For the first action (fundamental control of an outside device via a relay), we need an output port. To accept properly conditioned information from an outside source into the computer, we need an input port.

Let us put into words what we could do with both types of ports available to us. Suppose that our output port is controlling a relay which can turn a source of heat on or off. Further suppose that our input port is getting properly conditioned information that represents outside world temperature that is being warmed by the source of heat. We could then write a program in BASIC that would allow us to turn the heat on when the temperature dropped below a point that we had programmed as desirable and turn it off if the room temperature reached or exceeded the upper limit set by the program.

General Types of Ports

There are two common types of ports, latched and non-latched. Generally the output port is a latched (locked on) port and the input port is a non-latched port. When we say that a port is latched, we mean that when information is entered into the port, there is a device that holds or remembers the information put there. You can consider that the latch is a type of memory device which when told to turn on, stays on, and when told (by the program in the computer) to stay off, stays off.

How Many Ports are Available?

Let us consider the output port situation first. The normal way to configure or send information to an output port is via the data bus of the computer. This data bus is eight bits wide, D0 to D7. If you know a bit about binary or hex notation you know that the maximum decimal value that is represented by eight bits is 255. This means that you have a potential of 256 output ports that your computer could control. You have to remember that zero is a very real thing and that an output port number 0 does exist, making the total a very real 256 available output ports.

An output port is a slightly different ball game as you have two methods of selecting input ports. If you use a technique of memory-mapped ports (which will not be considered here), you can have literally thousands of input ports. If you use the same idea that is going to be presented for the output ports (a method called direct addressing) and limit yourself to the same eight bit concept, then you will again be limited to 256 input ports, which for our purposes should hardly represent a limitation of any consequence.

Starting Out

Using broad brush strokes, the information that we need to control our choice of input port comes from the low order eight bits of the address bus, A-0 to A-7 inclusive. We also need the help of the IN signal supplied by the computer. To select the desired output port, we need eight bits of help from the data bus, D0 through D7 inclusive, and the help of the OUT signal supplied by the computer.

We will also need one more available signal from the computer, which is the SYSRESET. The use of this and the other signals will become clear when we examine the intimate details of a practical I/O (input/output) device that will serve you well until you feel the need for a fancier device. This is not to imply that it is of limited utility; what limits it is your imagination and perhaps the cost of hanging things on the many potentially available inputs and outputs.

The rear of your TRS-80 has a forty-pin male edge connection portion that will mate with the proper socket. You can be fancy and get a prepared connector complete with attached ribbon cable on one end and bare on the other. You can get fancier (meaning more expensive) and get the same type of cable with connectors on both ends. Finally, you can go austere and get a wire-wrap plug with no cable attached and roll your own. The bottom line on costs (depending on how fancy) ranges from $4 to $16.

Methods of Construction

I prefer the perforated boards with the 0.1 inch spaced holes such as Radio Shack #276-1394. This board is 11.2 by 15.4 cm, large enough to work with and small enough to avoid the temptation of crowding. As it will not all fit on one board, you will be forced to go with the modular concept. This has its good points. You will change with the times as your knowledge and experience expand. The modular approach will allow you to re-use existing portions of this proposed project in the future, as whim dictates. For example, one of the modular divisions is to put all the relays and associated drivers on one board. You can make your own divisions in terms of modularity as they make sense to you and your knowledge of future needs. The five-volt regulated power supply is another logical modular division. Incidentally, a

suitable power supply for this project will *not* be detailed as this information is so readily available to you.

Where to Find the Signals

If your Level II manual is like mine, the information is not there. The information is available in the Level I book and better yet, in the *TRS-80 Micro Computer Technical Reference Handbook*, along with prints of the computer and excellent text on how the various portions of the computer function. This book is a worthwhile investment to read, *before* you start thinking seriously about tackling this business of interfacing. The key here is to find out how the TRS-80 works rather than how you *think* it works.

About Decoding

Our first premise is that we are going to control what our interface does by using BASIC language programming. Certainly we can use assembly language, but we have to start somewhere. There are two fundamental commands in BASIC that we will be using. The first (which controls the output port) is OUT. You can demonstrate this use very nicely by having it make the one output port that exists in the TRS-80 function. Here is the way we write the command in BASIC.

```
10 OUT 255,4
20 FOR T = 1 TO 300:NEXT T
30 OUT 255,0
```

Now press PLAY on your cassette machine and run the program. The motor will start and then stop. Your available output port, which is addressed as 255 (decimal), has turned the cassette control relay on and then off as it followed your BASIC program. There is a corresponding input port availability which has seen use by at least one commercial light pen designed for use with the TRS-80, but as a general concept the I/O abilities of the computer demand an external interface to be of real value.

When we instructed the computer to OUT 255,4, two things of importance happened inside the engine room. The 255 appeared on the low eight bits of the address bus. It made its appearance in binary form, which is 11111111, and the equivalent of all those ones or high states in digital terms made something happen. The other part of the expression OUT 255,4, namely the 4, caused the data bus to output on its low order 4 bits (D0 to D3 inclusive) which resulted in the appearance of binary pattern 0100 on these data bus terminals. The combination of these two actions made it possible for the computer to turn on the cassette motor, with further action of the OUT 255,0 instruction causing it to turn off when the delay loop had run its course.

The address bus is decoded or made to respond to the 255 address with the help of an eight input NAND gate. When *all* inputs of the NAND gate are high, we get a desired signal from its output. At the same time, the OUT instruction causes the OUT signal generated by the computer to go low. These two signals arriving at the right time complete the address decoding portion of the process. The binary pattern that appears on the data bus (due to the 4 in the expression in the program) is applied to a four-bit latch which by electronic manipulation caused the relay to close. When the timing loop ran out, the next expression (same port number of 255 but with the 0 in the expression) again called port 255; however, this time the data bus carried the binary pattern 0000 and the relay turned off.

Some Specifications

The actual unit to be described will have the following good features built in: There will be one input port that is eight bits wide; there will be one output port (latched) that is also eight bits wide. There is also an option to control fifteen relays. This variety of control will be enough to keep you busy well into the future. By some simple modification we can cut down on the number of relays controlled and add either more input ports or more output ports. More on this when we get to the nitty-gritty. The power requirements are easily handled by an LM 309 TO-3 style regulator with a minimal heat sink. The particular integrated circuits that show up on the schematic were used as they are relatively inexpensive.

A Brief Circuit Description

If you examine the schematic in Figure 1, you will see my first module. This consists of five integrated circuits.

1-74LS20 dual four input NAND gate package
1-74154 one of 16 data distributor
1-4050 CMOS buffer (hex)
2-8212 eight bit I/O port

The total current drain of this module is about 160 to 180 milliamps. The 8212 ICs will definitely run warm to the touch, which is normal. The 74154 will exhibit a bit of temperature above ambient and the remaining two ICs will show no apparent heat rise.

The one half of the 74LS20 marked A has its four inputs connected to the four address bus bits A-7 through A-4. We get a desired output from pin 6 *only* when all four input bits are ones. The 4050 CMOS buffer has four of its inputs connected to the bottom four address bus bits, A-3 to A-0. The other two inputs of this IC carry the IN signal and the OUT signal.

If address bus bits A-7 through A-4 are all ones and the lower four address bits are all zeros, then the address bus will carry a bit pattern in binary of

11110000, which translated into decimal is 240. This represents the lowest port number that we have available to us as an output port. If all eight bits were ones, then the binary pattern would be 11111111, which in decimal is 255. Thus we have some sixteen (16) possible port addresses at our command. The next chip in line is the 74154. It has sixteen output lines, any one of which may be selected by the combination of bits applied to pins 20, 21, 22, and 23. Note that these pins are fed signals from the low four bits of the address bus through the 4050 buffer. All of these output lines are normally high. When a given output line is selected, it goes low. Two sets of signals must be present for the 74154 to operate as desired. The proper bit pattern must be on the address bus *and* the signals to pins 18 and 19 must go low. One of these signals is from pin 6 of the 74LS20 and is used as a decoder. The other is the out signal which the computer generates as required in response to your use of the OUT command in BASIC.

The output line of pin 1 of the 74154 will go low when the bit pattern applied to the decoding inputs A B C D are all low or zeros. The output of pin 6 of the 74LS20 will go low only when all four of its inputs see a high state or all ones. Thus the binary pattern needed to make the 74154 output line go low is 11110000 binary or 240 decimal; thus our output port has an address of 240. The remaining output pins of the 74154 may be selected by addresses 241 to 255 inclusive. These output lines can be used as relay drive signals with some added circuitry.

8212 as an Output Port

The 8212 is an eight-bit wide I/O port that is tri-state capable. It contains eight buffer amplifiers along with eight latches and some circuit selection logic to make use of these major features. The eight inputs of the 8212 are connected to the eight bits that make up the data bus. The eight outputs are brought out to a terminal strip for ease of use. Notice that the SYSRESET line from the computer goes to pin 14 of this device. When you power up the computer, this signal momentarily goes low, which puts all the latches in the 8212 into a known state.

For the 8212 to output information in latched form, it needs two signals which are applied to pins 1 and 13. The second half of the 74LS20 provides one needed signal from pin 8. The other signal comes from pin 1 of the 74154. You can see that unless the output address of 240 is programmed, this latter signal will not exist and the port will not be called or selected.

8212 as an Input Port

This use is quite different from the output port setup. First, we do not use the latches at all, so they are disabled and made transparent. The 8212 has been turned into the equivalent of an ordinary tri-state gated buffer. Note

that pins 11, 13, and 14 are all tied together and then returned to +5 volts through a 1k resistor. The biggest single difference is that the only decoding, if you can call it that, is provided by the use of the in signal to gate the buffer amplifiers in this 8212 on or off as called for in the program controlling the input port.

Thus while you have to address the out ports (either the eight bit wide or the other fifteen available potential ports), the input port can be called by a dummy argument ranging between zero and 255. To program the output port, you need two items in the statement. You need the port address followed by a decimal number that will put the desired binary bit combination on the port output. For example: OUT 240,255 energizes our eight-bit out port and puts the binary pattern 11111111 at the port outputs.

To program the input port, we need the following type of statement: PRINT INP(X) where X is any letter or integer between zero and 255. This dummy argument for the input statement is needed to satisfy the TRS-80 syntax. The decoding scheme does not require the use of the dummy argument, but the computer protocol does.

A slightly different form of the input statement will allow you to assign the value existing at the input port to a variable, thusly: A = INP(X) where X is the dummy argument. Run the following program on your TRS-80 to see what happens.

```
10 FOR X = 0 TO 255
20 PRINT INP(X)
30 NEXT X
```

When you run the program, you will see a series of figures, all of them being 255 except the very last one, which will be 127. The program asks the computer to examine the bit pattern of all possible 256 port positions. It tells you that on all but the last one the bit pattern is 11111111, which is 255 decimal. The bit pattern for the last port is 01111111, or 127 in decimal.

Each time the computer issued an INP command, the in signal went low. The input port we are considering in the schematic needs only to have the IN signal (which comes to pin 1 of the 8212 from pin 15 of the 4050) go low. When this happens any digital information on the input port lines will be fed to the computer data bus. If the input lines of this port are just hanging in mid air (no connection), running the program just outlined would give an answer of 255. In this condition, all of the inputs are automatically in the high or one state. When I use the input port under discussion, I usually assign the dummy argument to match the number of the output port in use. This is merely a sort of mnemonic device on my part. You must, however, assign some number in the INP statement or the TRS-80 will give you an error message.

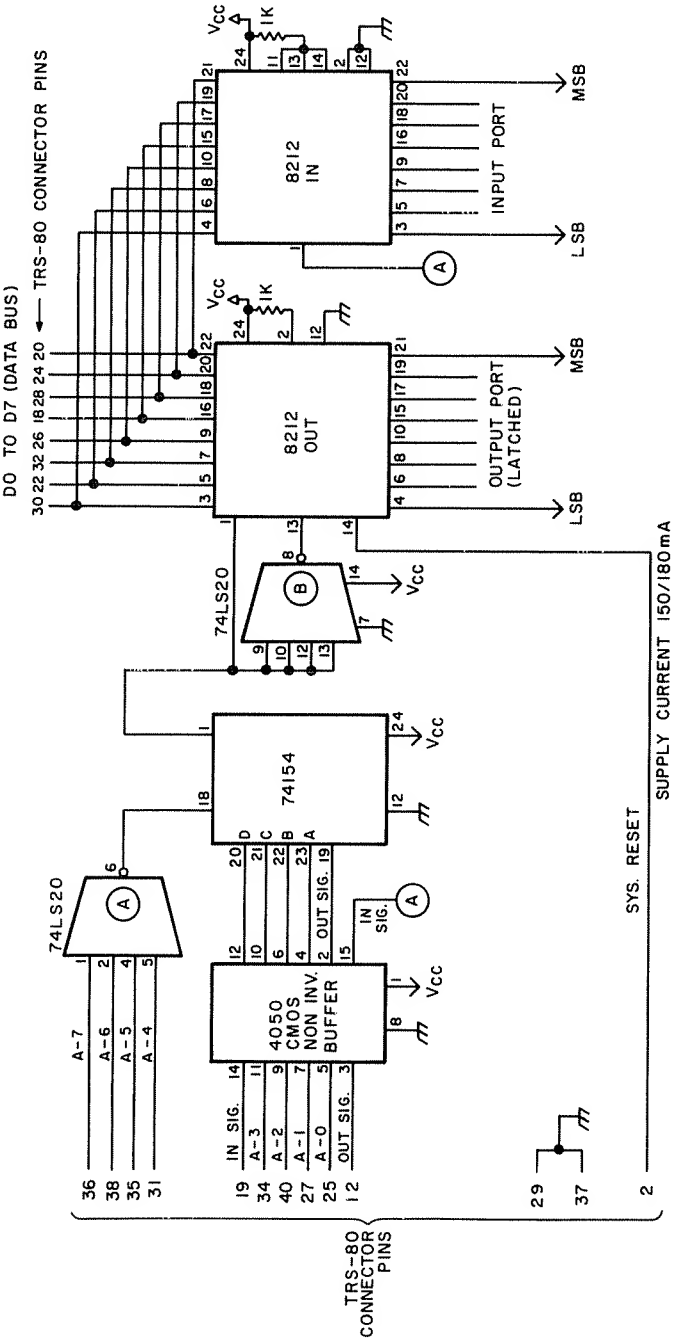

DO TO D7 (DATA BUS)
TRS-80 CONNECTOR PINS
30 22 32 26 18 28 24 20
3 5 7 9 16 18 20 22
8212 OUT
OUTPUT PORT (LATCHED)
MSB
LSB
8212 IN
INPUT PORT
MSB
LSB
Vcc
1K
74LS20
74154
4050 CMOS NON INV. BUFFER
OUT SIG.
IN SIG.
A-7
A-6
A-5
A-4
A-3
A-2
A-1
A-0
SYS. RESET
SUPPLY CURRENT 150/180 mA
TRS-80 CONNECTOR PINS
36 38 35 31
19 34 40 27 25 12
29 37
2

Figure 1

Power-Up Precaution

With the I/O port circuits attached to your TRS-80, when you power up the system, it is important to first apply power to the I/O device and then turn on the TRS-80. The reason for this is quite logical. When the I/O is first turned on, it produces glitches until it settles down. These unwanted power-up signals can find their way to the data bus, particularly from the input port, and mess up the TRS-80 initialization so that your computer may do weird things. This sad state of affairs can be avoided by turning on the I/O and letting things settle for a few seconds before you apply power to the TRS-80. You will not damage the computer but you can mess up its operating system. If you are a newcomer to I/O concepts, try to digest the signal flow a bit at a time. If you can build something *and* understand how it works, then you are ahead of the game.

Initial Testing

Assume that you have built the unit diagrammed in Figure 1 and have a five-volt regulated power supply ready to turn on. The first item of business is to apply power with a milliammeter in place to monitor the current drawn from the supply. Any drain up to about 200 milliamps is acceptable. If the current drain is in the ball park, let the unit cook for five or ten minutes.

Test the ICs for undue warmth, remembering that the 8212 chips will get from warm to rather warm in normal operation. The bulk of the current drain is going to these chips, hence the heating effect. If the unit has passed this smoke testing with flying colors, it is time to give the overall wiring one last visual check for solder splashes or bridges, making sure that all of the ICs are properly seated in their sockets, etc. Then, and only then, should you consider the initial connection of I/O to the TRS-80.

The first connection episode should be done with no power applied to the I/O board. Power up the computer and run any test program to check for proper computer operation. If the TRS-80 is functioning normally, turn it off and then apply power to the I/O unit. Now restore power to the TRS-80 and once again run a test program to assure yourself that some misadventure in wiring, inadvertently getting the edge connector plug upside down, etc, is not going to send you into a corner, muttering *mea culpa*. What we are trying to do is protect our TRS-80 with common sense which is worth several bushels of blown fuses.

Input Port Test Program

Type this program into the machine and run it:

```
10 PRINT INP(240);
20 GOTO 10
```

When you run the program, you will see the screen fill up with a series of 255 figures. This tells you that all eight input lines of the input port are high, which is normal. If you do not get this indication, shut down and check your wiring.

Assume that we have gotten the desired results. With the program running, take a test clip and short pin 3 of the 8212 to ground. You should now see the 255 change to 254 because you have changed the bit pattern from 11111111 to 11111110 (from decimal 255 to decimal 254). If you short the inputs one at a time (to ground), you will see the numbers on the screen change to reflect the altered bit pattern being fed to the data bus via your function input port. Pin 3 is the LSB or least significant bit and pin 22 carries the MSB or most significant bit. If you short pin 22 to ground, you should see the number on the screen change to 127, which represents 01111111. If your results match this format, then your IN port seems to be functioning properly.

Testing the Output Port

This will go more quickly as you are fairly sure you have no problem caused by the I/O being hooked to the computer.

Type in and run this test program:

```
10 OUT 240,0: FOR X = 1 TO 400:NEXT X
20 OUT 240,255: FOR X = 1 TO 400:NEXT X
30 GOTO 10
```

Run the program. Apply a voltmeter to the output port pins 4, 6, 8, 10, 15, 17, 19, and 21 in turn. You should see a dc reading that goes up and down in time with the two delay loops in the program. On the high swing, my Simpson 260 goes up to about 4.5 volts, but this will vary with the damping of your particular meter movement. What you are doing is alternately feeding all ones (highs) to the outputs when the port is on and then getting all zeros (low states) when the port is off, on and off being defined in the BASIC program. Remember the bit pattern for 255 is 11111111, and the bit pattern for zero is 00000000; hence the indications that you get are correct. Once again, if you do not get this result, check for wiring errors or defective ICs. My experience has been that if you have taken the precaution of obtaining your ICs from a reliable supplier, the usual cause of any problem is a wiring goof.

At this point, you may assume that the fifteen outputs of the 74154 will work as expected when we put them to use. Unless you have a logic probe, there is no practical way to check for a pulse output, as the pulse is just a bit better than one microsecond in duration.

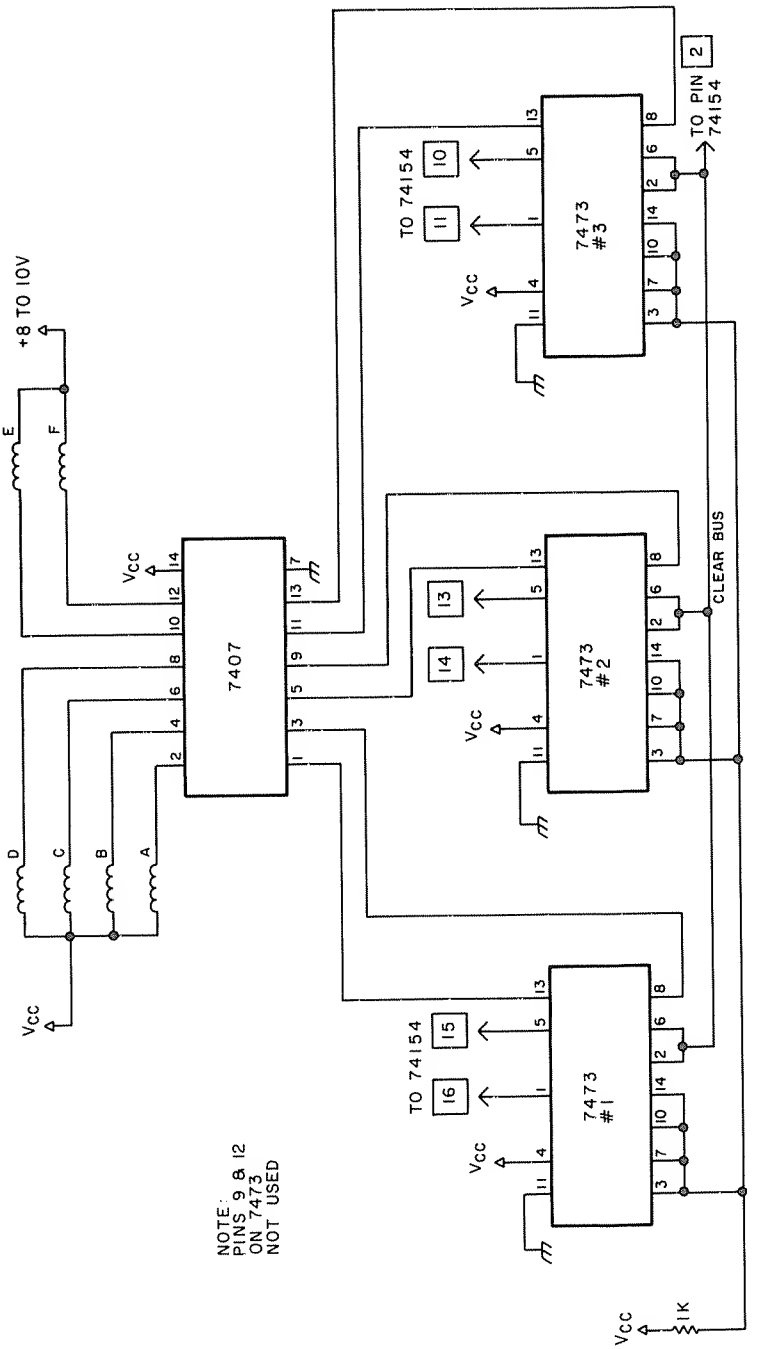

+8 TO 10V
E
F
D
C
B
A
Vcc
7407
7473 #3
7473 #2
7473 #1
TO 74154
TO PIN 2 74154
CLEAR BUS
1K
NOTE:
PINS 9 & 12
ON 7473
NOT USED

Figure 2

The Relay Module

We are now going to examine the module that allows us to turn relays on and off under program control, with the aid of module number one that we have just examined. There is a flexible side to this module. By this I mean you might wish to duplicate it or you might wish to expand it. Some possible areas for so doing will be touched upon as logic dictates. The schematic shown in Figure 2 shows what the beast looks like on paper.

General Description

There are three 7473 JK flip-flops and a 7407 hex buffer making up the chip complement. Please note that on the 7473 chips, pins 9 and 12 have no connections made to them. All of the J-K pins on the flip-flops are tied to Vcc through a common 1000-Ohm resistor. All of the CLEAR pins of the flip-flops are tied together and then this common lead goes to pin 2 of the 74154. We will examine just why a bit later. For now, remember that when the CLEAR goes low, the Q outputs of the flip-flops go low and the NOT Q outputs go high. The NOT Q outputs are the working outputs of the flip-flops that go to inputs of the 7407 hex buffer. The outputs of the 7407 act as relay drivers; in this case I have six relays that I can control at will by proper computer programming.

Getting Down to Business

You recall that, due to the decoding available from module one, we have sixteen addresses that we can summon to do very specific things. As set up for this adventure, address 240 activates the eight bit output port. The balance of the addresses (241 to 255), when called, make each succeeding output pin of the 74154 go low momentarily, after which it returns to its normal high state. Thus when we call one of these addresses, we are really only generating an output pulse from the selected OUT pin of the 74154. What we make those pulses (or strobes) do separates the sheep from the goats.

Since we are dealing with relays, it would be nice to know (at power up) what the contacts are doing. We want to be able to assume that the contacts are open, or, if you prefer, closed, before we do any programming. On power up, there is no guarantee that the flip-flops will come up as you desire them. In my case I want all of the Q sides of the flip-flops to be low so that all of the NOT Q sides will be high. Why this is so lies up ahead.

Certainly, we *could* have used the system RESET signal of the TRS-80 which was used in module one. Why this was not done is a fair question. Here we can be flexible—we have the facility to control more than a single module two. My preference is to have independent control of RESET for each module two that I have in the system, something that use of the TRS-80

SYSRESET signal would not readily give me as it exists for use *only* when the TRS-80 is first powered up. This is why the CLEAR bus of module two goes to pin 2 of the 74154. I can make this pin go low momentarily by calling its port address, which is 241, and thereby set the contacts of all six relays to known starting positions.

Please note that once again, due to the nature of the decoding used in module one, we run into the use of a dummy argument in the form of the OUT expression when calling any of these relay control ports. To use the RESET feature that I have built into port address 241, you would program: OUT 241,X where X is any number from zero to 255. Once again, I generally use the port number for the dummy argument as a memory convenience.

Buffer Relay Driver

My unit has a mix of relays. Four of them are reed relays that operate at ordinary TTL circuit voltages. These are shown connected to 7407 buffer output pins 2, 4, 6, and 8. The maker has packed four reed relays into a very tiny package, each relay having one set of normally open contacts—light duty contacts, I might add, similar to the cassette relay in your TRS-80.

The other two relays connected to buffer output pins 10 and 12 are, by reed relay comparison, quite substantial in current carrying capacity. They have 2500-Ohm coils and pull in quite a supply source of voltage. They are the large crystal can type and have double-pole, double-throw contacts. It is obvious that with these six relays you have a good deal of control of outside devices, through computer programming control. At first, it may seem that using the relays that take other than standard TTL working voltages means building a separate power source, but think a bit. You need a regulated five-volt supply to power this whole interface, and any regulator needs to be fed from an unregulated source that is higher than the desired regulated value. It is that voltage source that can feed the two larger relays.

My six relays are operated by calling addresses 249 to 254. Notice that I avoided using 255 as that is assigned to your cassette relay. For the record, address 255 is available at pin 17 of the 74154, which is nice to know if your cassette relay has died or is showing signs of heading west from "welditis of the contacts."

The following assumptions are made. Modules one and two are tied together and turned on. Then the TRS-80 is powered up, this order being followed to avoid glitching the computer's power-up programming routine.

The first thing we want to do is get the relay module relays in our standard desired postion; in my case I want to make sure that all the reed relays have their contacts open. To do so, I run OUT 241,241 and pin 2 of the 74154 goes low long enough to make all of the NOT Q outputs of the flip-flops high. The reason for this being a desired condition is worth examining. The 7407 is a

power buffer and is non-inverting. The same state you put into it (high or low) comes out of it. If you put a low into it, the output pin goes to ground. This would effectively put the relay from its supply voltage to ground, and the relay would turn on. The opposite is true if a high goes into the input of the 7407 buffer (being used here as a relay driver). Now you can understand that I have specified my starting base for relay contact positions being open and how this comes about by the use of the RESET or CLEAR pulse from pin 2 of the 74154.

Now that we have our relays where we want them, let us consider a simple program to open and close a given set of relay contacts. We will use an ohmmeter as an indicator and for specifics we will use the reed relay marked A in the diagram. This relay is connected to pin 2 (output) of the 7407. The input to this amplifier is pin 1, which is in turn connected to pin 13 of its controlling flip-flop (NOT Q). The final link in the chain is pin one of this flip-flop which gets its control pulse from pin 16 of the 74154. Pin 16 of the 74154 will go low when its port address of 254 is called by your program.

Remember our fundamental assumptions. We have gotten our relay contacts into a known starting position and our ohmmeter is across the contacts of relay A. If things are working as planned, the meter shows no indication as the contacts should be OPEN.

Now we can insert this little program into the computer and RUN.

```
10 OUT 254,254:REM remember the dummy argument
20 FOR X = 1 TO 400:NEXT X:REM Timing loop
30 GOTO 10
```

When you run this program, the meter will pulse with the opening and the closing of the contacts. How can the same command act to both open and close the relay contacts? The answer is that we are using a J-K flip-flop. This IC will change state every time that its clock input is presented with a properly shaped pulse that makes a high to low transition. If you are at all familiar with the conventional binary divider, then you can see what is happening.

If you take a head count of what we have used out of the sixteen available addresses you will see that:

- We have used address 240 to control the eight-bit output port.
- We have used address 241 to generate the CLEAR signal for module two (relay module).
- We have used six more addresses (249 to 254) for individual control of the six relays. We have eight available control ports for use, as your imagination dictates.

The 7407 can accept up to 30 volts on the output pins; *but* pin 14, which is the supply pin for the insides of the chip, should be limited to the standard five volts. I have shown no inductive kick diodes across the relay coils and

the unit functions well without problems. You may wish to install them as a precaution across each relay coil (cathode to the supply side of the relay and anode to the side of the relay going to the IC output pin) if it suits your needs and the particular relays that you use. If I had used heavy 12-volt relays with coil currents of 15 to 20 milliamps, then I would have felt safer with the diodes across the coils.

Incorporating the use of these relays is simple, as you can see how little programming it takes to get them to turn on or off. If someone makes a tri-state relay, then we have a bit of a problem. See Table 1 for a table of port addresses and the output pins of the 74154.

PORT ADDRESS	74154 OUTPUT PIN
240	1
241	2
242	3
243	4
244	5
245	6
246	7
247	8
248	9
249	10
250	11
251	13
252	14
253	15
254	16
255	17

Table 1

TUTORIAL

Into the 80s

Into the 80s
Part I

by Ian R. Sinclair

Many articles are written for those of you who have owned a TRS-80 for some time and know what programming is all about. "Into the 80s" is dedicated to the thousands who are just about to buy a TRS-80—or have just bought one—and want to know what they've gotten into.

Throughout the series, we'll be talking about the Model I Level II TRS-80. Level is Tandy's word to indicate the complexity of programming language its computer is able to understand. We're dealing with the Level II only for two good reasons. First, Level II is much more useful. And, secondly, the Level I computer is already accompanied by an excellent manual for the beginner.

One measure of a computer's power is its memory. Memory dictates the amount of information (data) a computer can store and is measured in units called kilobytes, shortened to K. The 4K computer can store four kilobytes of data. Nothing in this series will cause you to run out of memory space on a 4K TRS-80.

Before you proceed, be assured of one point. The computer does exactly as you instruct it—nothing less, nothing more. If you have not put in the instructions to print letters or numbers on the video screen, then these letters or numbers just don't get printed. One of the humiliating things about being a computer owner is knowing that whatever goes wrong is your fault.

Practical Pointers

Take a look at the sockets at the back of your 80. They are European DIN-type sockets and match five-pin plugs. There are three of them: one each for power in (from the transformer unit), cassette in/out (I/O), and video out.

Before you start using your 80, take the advice of an old hand and label these plugs and sockets with differently colored tapes. I use red for the power plug, green for video, and yellow for the cassette. If your 80 sits in the same place and you never unplug it, this isn't significant. The odds are, however, that some day you'll want to shift it, and you could easily end with plugs in the wrong sockets, since they are identical and easily confused in poor lighting conditions.

Keeping your system cool is another useful tip. Try not to have a desk lamp shining on the keyboard, for instance. It doesn't help, either, if you've used the computer all day with bright sunlight heating up the

keyboard casing and the electronics inside. High temperatures also damage cassette recordings.

I've found that a normal room temperature of 70-75 degrees won't cause the TRS-80 any distress, even if you use it all day. It's another story if you have an expansion unit attached, but we won't go into that. Just make sure that that little black box which is the transformer unit is on the bench or on the floor, with room for air to circulate around it; don't put it inside a box or surround it with books.

Power It Up

You're ready to power up. Plug all the line plugs into the wall sockets and switch on. *Don't* switch on the keyboard first—always have power on the cables before you turn on the units, because the switch for the keyboard does more than just switch power.

Start the countdown by switching on the monitor. Let the monitor warm up for a minute, then switch on the computer keyboard. The ON/OFF switch is at the back, next to where the power supply plug enters the casing. It's deliberately made a bit hard to find, because when you switch off a computer, all the program material you had stored in it is lost, gone forever, unless you recorded it on a cassette previously.

As you press the ON/OFF switch, you'll see the video screen suddenly filled with a mixture of numbers, letters, and odd shapes. That's "garbage," caused by the computer memory being activated. Each little cell of memory can store a bit of information, zero voltage represented by a zero, or +5 V represented by a 1.

Upon power-up, when all of these cells are activated, some come on as 1s, some as 0s. About 8192 of these memory cells send signals to the video screen. The cells which are set to 1 cause parts of the video screen to light up, and the cells which are left to 0 keep the screen dark.

The result is a display of light and dark pieces at random or almost at random. Circuits inside the computer force these light and dark places into patterns, the patterns which we call letters, numbers, and graphics blocks, and this is the pattern we see just as we depress the ON/OFF switch. When you release the power switch, the garbage clears, because the switch-on starts a memory clear routine for the video screen and memory—that's why it isn't a good idea to switch the keyboard on before plugging in the line and powering up.

The Mystery Message

Shining on the video screen in all its glory is the message MEMORY SIZE? It has floored many a beginner. Did no one back at Fort Worth tell

the machine what its memory size is, you ask? Ignore it for the moment, press the ENTER key, and these more reassuring words appear:

```
RADIO SHACK LEVEL II BASIC
READY
>
```

Your computer is prepared for programming in the BASIC program language.

What about the MEMORY SIZE? message? Well, as it happens, a lot of computer tasks can be performed faster and more efficiently by giving instructions directly to the microprocessor chip inside the computer. This chip needs special instructions, called machine code or object code, and these instructions can't be loaded into the computer in the same way as an ordinary BASIC program.

Unless you're going to use machine-code programs right away, though, you can ignore the MEMORY SIZE? question.

The one time you can't ignore it is when it appears while you're running a program. When that happens, it's an unwelcome sign, called a re-boot, that something is very wrong with your program. The computer has started its power-on sequence again. There's no harm in it, but you will have lost your program unless you saved it on cassette earlier.

The READY signal is an invitation, but unless you know what it's inviting you to do, you can't take advantage of it. At the READY signal, you can either load a program from a cassette or you can type one yourself. Since you'll learn more about the 80 from writing your own programs, however simple, we'll start there and leave the frustrations of cassette loading for a later date.

READY is an invitation for a BASIC program. BASIC is an acronym for Beginners' All-purpose Symbolic Instruction Code, and it's the easiest of programming languages to learn.

Why should we have to learn BASIC?

It's all bound up with the way computers work and are designed. The fastest and most efficient programs are written in machine code, but learning and using machine code is a painful business, and writing machine code is a frustrating experience.

For these reasons, computer designers have continually sought to make it possible to give instructions in simpler forms, using English words (or Spanish, French, Italian, and others) and stringing them together in a way which is reasonably simple to understand.

If you've looked at some of the programs printed in the back of the TRS-80 manual or published in *80 Microcomputing*, you might not quite believe this last statement, but compared to most other computer

languages, BASIC is reasonably easy to understand. We can devise simpler languages, but the penalty for using a simpler language is either that it doesn't do as much as we would like, needs more memory, or takes longer to run. Right now, your TRS-80 comes with its BASIC language built in.

Like any other language (and I've had to cope with Latin, French, and Greek in my time), BASIC is best learned by using it. Unless you intend to use your computer simply to run programs written by other people and obtainable on cassette, you haven't much choice—learn BASIC!

Unlike other computer languages, BASIC isn't standardized. A program written in BASIC for another computer may not run on the TRS-80, and vice versa, unless you make a few changes.

It is possible to write BASIC programs that will run on any computer equipped with the BASIC language (the Adam Osborne programs are good examples of this), but you can get a lot more out of your 80 if you know the peculiarities of its particular dialect.

This dialect, incidentally, is one of the most advanced BASICs fitted to a small computer. We can't hope to show all the features of BASIC programming in this series, but we can try to fill the gap between elementary BASIC textbooks and Radio Shack's *Level II Reference Manual* that comes with your computer.

Program Proverbs

If you've never programmed a computer before, learning BASIC on your own can create as many ulcers as guarding a bank during a revolution. Computers are fussy about the way you use the language. If the manual says that a word must be followed by a comma, then it really must be followed by a comma, or your program won't run.

If, on the other hand, a word needs a semicolon after it, you can't get by with just a comma or a colon. This punctuation is translated into instructions by the computer, and different marks denote different instructions.

Translated? That's just what happens. There are about twelve thousand bytes of memory inside your TRS-80 which you can't alter. The professionals call it ROM, or read-only memory. It reads your BASIC program and converts each letter into machine-code instructions to the microprocessor. Because each instruction is converted and then carried out, the instructions are much more long-winded than those in a machine-code program.

Bigger computers can do what's called compiling, which means translating the entire program into machine code in one run, and then running the machine code. The TRS-80, like all small computers, only interprets—it converts each instruction in the BASIC program into code, runs it, and then goes on to the next part. The difference is between dictating

directly to a secretary, with all the ums and ers of speech, and delivering an edited tape to an audio typist.

Let's start again with the READY signal staring us in the face. Your first response toward writing a program is to type a line number. The line number is a tag we can attach to an instruction so that we and the computer know where to find that instruction.

Take any number between 1 and 65529. Normally, you start at 10 or 100. It doesn't matter what numbers you choose, since you don't use more memory by numbering lines 100, 200, 300 than by numbering them 10, 20, 30. It is important to remember that the program should flow from the lowest numbered line to the highest.

Since we have to start somewhere, type 10 and a space. Your program now starts at the line numbered 10. Since it's a bit depressing to stare at a blank screen, we'll get the computer to PRINT my name.

```
10 PRINT "IAN R. SINCLAIR"
```

Remember, the quote marks must be in place, because they indicate to the computer that the words between them are to be printed on the video screen and are not part of an instruction. Marks that divide one kind of word from another are called delimiters, and the quote marks are the easiest of these delimiters to work with. Make sure that anything you want printed to the video screen starts and ends with quote marks.

Made a mistake? We're not all trained typists, so the TRS-80 is very forgiving. Use the backspace, the arrow which points to the left, on the key next to the @. Each time you press this key, one letter of your instruction is wiped out, and you can type another one. Keep on until the line is perfectly typed.

Now that you've typed the line, you must press ENTER to make certain that it's planted into the computer's memory. Even if you forget the line number, you still get a result—the words which were inside the quote marks will be printed, but you don't have a program because it won't repeat, and you must start again. If your line was entered correctly, it is now a very simple program. We can run the program by typing the word RUN and then hitting ENTER again.

All right, so it's not impressive, but each time you type the word RUN and hit ENTER, you'll get my name printed on the screen. Makes *me* feel good at least.

Try typing the instruction with no space between the 10 and the PRINT, so that it reads 10PRINT "IAN R. SINCLAIR". Does it make any difference in the way the program runs? Did I say that computers were fussy?

Now type LIST10 and hit ENTER. This prints your instruction line on the screen again. Notice anything about it? Right, the space has been put in again between the 10 and the PRINT. You can't cheat your TRS-80 this way!

Now try some more deliberate errors. It's just as well to know at this stage what effects they have, rather than being tangled up with an unfamiliar error message later on. For a start, leave out the first set of quote marks. Then try typing PPRINT or PRIINT instead of PRINT. Doesn't run, does it?

Instead, you get the words SN ERROR. SN is a shortened version of SYNTAX, a word that language teachers use to mean the way in which a language is constructed. If you say in English, "I am here since yesterday," your syntax in English is poor, but the same phrase in French is grammatically correct.

Syntax is a matter of rules, and an SN error means you have broken a rule of BASIC by leaving out a delimiter or misspelling an instruction word. Some newer computers, incidentally, reject a syntax error the moment it is entered, but the TRS-80, like its generation of home computers, doesn't spot the error until you try to RUN.

How much can you PRINT in one instruction? Try it! Type 20 PRINT "... and then type as many words as you like. You'll find that the words don't run off the edge of the video screen, but form new lines of text on the monitor. The video monitor accepts only 64 characters/ line, while the computer allows up to 250 characters/line. The computer no longer responds to keyboard input when the 250 character limit is reached.

PRINT Plus Spaces

Now try the one-liner below:

```
10 PRINT" SPACED ALONG"
```

Because we've labeled this line as line 10, it has wiped out the old line 10. Notice that the space between the first set of quote marks and the first letter of the word affects the way it prints. This is one way of adding spaces.

Another way is by indicating tabs. Tab means the same as it does on a typewriter: tabulation. The width of the video screen is divided into 64 starting points, numbered from 0 to 63. Using TAB( ) selects one of these as a starting point for what you print.

```
10 PRINT TAB(25)"IN THE CENTER"
```

Notice the syntax—parentheses after TAB enclose the number, between 0 and 63. The quote marks surround the words to be printed.

If you had put the first set of quote marks between PRINT and TAB, you would have printed, at the left-hand side of the screen, the phrase TAB(25) IN THE CENTER, which isn't exactly what we wanted.

Why TAB(25)? Well, I counted the letters and spaces in IN THE CENTER and made it 13 characters. I rounded that to an even number, 14, then subtracted from 64, leaving 50. Half of 50 is 25, and that's the tab number. Why? Well, the number of letters and spaces gives the number of

positions on the line which are used to print. If we want these words to be centered, then there has to be an equal number of spaces on each side, and we find this by dividing the number of unused places, 50 in this example, by two.

Try printing your own name centered on the screen.

Notice, by the way, that we're still entering each one-line program as line 10, deleting the previous program in line 10. Later on, we'll look at other ways of removing old programs, getting rid of unwanted lines, or running only the lines we want.

TAB is one of the BASIC commands or functions that not all small computers have, but it's only the start of the options which are available to the TRS-80 owner.

```
10 PRINT "SPACE", "THE", "WORDS", "OUT"
```

Type the above line very carefully. See the commas? Each comma lies outside the quote marks that set off the words, because we don't want the commas printed. If we type:

```
10 PRINT "SPACE, THE, WORDS, OUT"
```

hit ENTER, type RUN, and hit ENTER again, the printout on the video screen would be:

```
SPACE, THE, WORDS, OUT
```

As it is written, the effect is quite different, as you will see. The commas have commanded the computer to space the four words across the width of the screen. Each comma instructs the computer to start printing the next word at the next print zone. There are four print zones, each of which can take up to sixteen characters (letters, numbers, or spaces). If you PRINT more than sixteen characters in a zone, the comma causes a skip to the next zone. Try the following program.

```
10 PRINT "THE ZONES WILL TAKE",
"SIXTEEN CHARACTERS EACH"
```

The comma is used as a print delimiter. We can also use the semicolon as a delimiter, but with a very different action. Try entering the program in Example 1. With more than one line of program, you must remember to push ENTER at the end of each typed program line. When you run the program (type RUN and hit ENTER), what happens?

The semicolon is another signal to the computer when it's used in this way. Typed after a printed quantity, the semicolon means: Use the same line and keep printing, so that the words we typed in three separate instruction lines end up on one single line of type. This would happen, incidentally, even if we had several other lines of instructions between these PRINT instructions, so watch carefully for these semicolons if you are entering a program which has been written in BASIC by someone else. All small computers use this form of instruction.

What happens if you leave out the semicolons? Try it! Each PRINT command causes the video to start on a new line. This is one way you can space out your printing vertically.

Meanwhile, look at Example 2 for a very powerful command which few small computers have in their BASIC. It's the POS function. POS means position, and it means the position of the cursor mark, that short line which shows where the next letter will appear on the screen when you type a program line.

As a result of the POS instruction, the computer makes a note of how far along a line you have printed. Add five to that place number, as we've done in Example 2, and the result is five spaces between each word.

When you type line 10, by the way, don't hit ENTER until the end of the instruction after the last set of quote marks. If you hit ENTER before then, you're indicating that you want to start another numbered instruction line. Another point is that you should type DELETE 20-30 and hit ENTER before running the program. If you don't, you'll print the words in lines 20 and 30 of the previous program, unless, of course, you have switched off between working Examples 1 and 2.

By this time, your video screen must be looking a bit cluttered, so let's look at another useful function that helps clear things up. Example 3 shows a four-line program (we're getting more adventurous) which makes your video printouts look better. The new instruction, CLS, means, simply, clear the screen.

When the program is run, the screen clears, removing the program lines. After the CLS instruction has been used, the next PRINT instruction will place the words in quote marks on the top line.

In line 30, I've used the word PRINT by itself. What does that do? Try leaving it out by typing DELETE 30 and hitting ENTER. Then type RUN and hit ENTER again. See the difference? The PRINT command in line 30 causes a one-line space between HEADING and the words in line 40. Want a two-line space? Then type 30 PRINT, hit ENTER, type 40 PRINT and hit ENTER again. Now run this one and watch the larger gap appear between the heading and the line of words.

Time to take a look at the program. Type LIST and hit ENTER. Your program appears under the last lot of printing, with the lines in correct order. That's just one example of why the BASIC language uses these line numbers. We can place any line number against a line, and the computer sorts them into order. If we use the same number for two different lines, then the last one typed and entered wins, the older one is deleted. We can also delete lines by typing DELETE, then the line number, and hitting ENTER.

Can we delete more than one line at a time? Sure we can. Just type DELETE 10-40, and every line of the program in Program Listing 1 will be rubbed out. The other way we can remove a whole program, but this time without needing to know how many lines it has or what their numbers are, is to type NEW and then hit ENTER.

More Spaces

Suppose we want to print a word at the center of the screen. There are sixteen lines of print on a full screen, so we could print on line eight. We could type seven lines with PRINT, but no words, to make the print position move down one line at a time. We could then use the commands PRINT TAB( ) to space the word to the center of line eight.

There's a much easier way of doing this sort of thing with a TRS-80, by using the PRINT@ command. It must be entered correctly, with no space between the T of PRINT and the @ sign, and a comma immediately after the number following @. If you put an unwanted space in, you'll get the SN error message when you try to run it. A much less obvious error is typing @ with the SHIFT key depressed. If you hit SHIFT and @ at the same time, the @ appears as usual on the video screen, but the code number which is fed into the computer is NOT the correct one. You'll get the SN error report when you try to run it, but the line will look good on the screen.

With these warnings in mind, try the program in Example 4. Remove the previous programs by typing NEW and entering. Now type in the three lines of the new program and run it. Interesting? The word GREETINGS appears around the center of the screen, and the next line of print is four lines under that.

Take a look at page E1 at the back of your Level II manual. Turn the page so that the numbers are all right side up, and you can read off the PRINT@ numbers and the TAB numbers. The numbers in heavy type at the top are the TAB numbers for each line, and also the PRINT@ numbers for the first line, the top line. For the second line of PRINT@, the numbers start at 64 (see the columns down the left- and right-hand sides), for the third line, 128, and so on.

To find a PRINT@ starting number for any position on the screen, pick your spot, locate the TAB number at the top of the page and the PRINT@ number for the start (at the left- or right-hand side), then add the two. For example, if you want to start around the center, try TAB(31) on the line starting at 448. That gives us a PRINT@ on the number 448 + 31 or 479, so we type PRINT@479, then add the quote marks and the message. Remember the syntax: PRINT, no space, @, no space, number, no space, comma, then quote marks for the message, and keep your fingers off that shift key when you are typing the @.

Using More

The TRS-80's big, big BASIC allows us yet another way of printing which isn't available to people with other types of machines. The new instruction this time is PRINT USING, and it instructs the computer to arrange the printing to suit some definite pattern which must be specified in the PRINT USING command. PRINT USING is most useful when you have to print out a number in standard form, such as a price or a sum on a check. It's also useful when you want to round off a fraction.

Since the number of times you are likely to want to use this command in your own programs is limited, compared to the everyday ones such as TAB and PRINT@, we'll look at only a few of the PRINT USING commands.

Example 5 shows PRINT USING applied to rounding off a fraction. Enter the program and run it to see how the number is printed. This is a smart way of making sure that your printout doesn't contain lots of figures after the decimal point. After all, you wouldn't like to think that you had just printed a check for $56.2357.

Another useful feature of the PRINT USING command is that it can insert a floating dollar sign. Now, if you thought that the dollar was sinking rather than floating, let me explain that phrase. You might want to print out something like amounts of $1.50, $26.40, $147.50, and so on. What the floating dollar sign does is position itself ahead of the first figure of the number so that you don't print 1$27.50 and $02.40. Try it out with the program in Example 6.

The Level II manual has a large number of examples of PRINT USING, so we'll leave this one, which is a more specialized command than most of the ones we shall be using in this series.

Bugs

Depending on the age of your 80, you may already have met the dreaded kkeybbounce. You type PRINT and it appears on the video screen as PPRINT or PRIINT or some other weird combination of repeated letters.

When you try to run a program with an unwanted double letter in a command word, you'll get an SN error message, meaning that the internal circuits of your computer simply don't recognize the word. Of course, if you have an unwanted double letter in a message which is enclosed in quote marks, then it will simply be printed out that way.

Keybounce is a problem that plagues any mechanical switch, like keyboard switches. You press a key and the electrical contacts close. But, because they're made from springy material, they bounce open again before finally closing and staying closed. Each time the switch closes, it completes an electrical connection, and if that happens to be the electrical connection which prints the letter P on the video screen, then you get two of them.

Every manufacturer of computers gets around this by using a time delay each time a key is pressed. The computer takes no notice of the key until the time delay is over. Only a small time delay of about a thousandth of one second (one millisecond) is needed.

Radio Shack seems to have given short measure to this problem on the older TRS-80s. On some models, keybounce can be fixed by pulling off the keycaps and cleaning the contacts. That's what Radio Shack says, anyway, but my own TRS-80 has fixed keycaps that don't come off easily, and it bounces very badly. The keybounce is so bad, in fact, that if there were no cure for it, I would have scrapped the whole thing months ago.

Yes, there is a cure, and it works. Radio Shack supplies a program on a machine code tape entitled KBFIX. Enter this one, and your keybounce troubles are over until the next time you switch on. Incidentally, more recent TRS-80s have no trace of keybounce.

Sometimes keybounce is just a nuisance. Other times it can cause a complete hangup of your computer, and one of these other times is when you type LIST and get LLIST. LLIST is a command word which was unfortunately chosen for printing a list on the Radio Shack line printer. Run LLIST and the result is—nothing. No keys have any effect, nothing appears on the screen, and the computer appears to have died on you, with just that accusing word LLIST staring at you from the screen.

What's happened is that you have commanded the computer to print a list on the printer, and because there's no printer connected, it's waiting for you to connect one. Don't rush out with a fistful of dollars, because you can recover from this stall in two ways.

One is to switch off completely, but that way you'll lose any program you had in the computer. The easier way is to push the RESET button at the back of the computer, at the opposite end from the ON/OFF switch. On my 80, this is under a small flap which also houses the connector for expanding memory. Pushing and releasing this switch removes the hangup, and the computer is ready to run again. Whatever your manual says, you don't lose your BASIC program when you use RESET unless the Radio Shack expansion interface is connected.

The keybounce problem can also be avoided by using a short BASIC program, shown in Program Listing 1. It's a sight longer than the others we've used in this part, and you have to be sure that you've typed each character correctly. Run it, and it sorts out the keybounce and then deletes itself!

At this stage, it's not easy to explain how it works, because it makes use of parts of memory we don't normally use, the reserved RAM. The purpose of these memory parts is kept a close secret, and it's only when someone with a bit of time to spare investigates them that we even get to know about them.

Keybounce is just one example of what folks in the computing business call a bug—a flaw in a system. A bug may be in the operating system, or it may be in a program. Wherever it is, "Into the 80s" will show you how to stamp out some of the most active bugs.

```
10 PRINT"SEE THE EFFECT";
20 PRINT"OF A SEMICOLON";
30 PRINT"ON THE PRINTOUT"
```

Example 1

```
10 PRINT"SPACE";TAB(POS(0) + 5)"THE";TAB
   (POS(0) + 5)"WORDS";TAB(POS(0) + 5)"EVENLY"
```

Example 2

```
10 CLS
20 PRINT TAB(28)"HEADING"
30 PRINT
40 PRINT TAB(5)"THIS IS A NEATER WAY OF PRINTING"
```

Example 3

```
10 CLS
20 PRINT@475,"GREETINGS"
30 PRINT@704,"THIS USES THE PRINT@COMMAND TO SPACE LINES"
```

Example 4

```
10 CLS
20 PRINT USING"##.##";2.736
```

Example 5

```
10 CLS
20 PRINT USING"$$##.##";21.471
```

Example 6

```
1 POKE 16553,255:CLEAR:FOR N = 16480 TO 16492:READ K:
  POKE N,K:NEXT:FOR N = 16435TO16437:READ K:POKE N,K:
  NEXT:POKE 16405,0:DELETE 1-2
2 DATA 205,227,3,183,200,14,40,16,254,13,32,251,201,195,96,64
```

Program Listing 1

Into the 80s Part II

by Ian R. Sinclair

It doesn't take long for the novelty of printing your name on the video screen to wear off. There are more interesting ways of using the TRS-80, including the manipulating of variables.

A variable is a code for something, which might be your name, your driver's license number, or any other piece of information you choose. The fact that you can change this code at any time makes it variable, but once you've defined it, the computer will make use of this code any time you instruct it to.

String Variables

Using the methods you learned in the last section, type ENTER and run the program in Program Listing 1. There are a few new points to make here. First is the use of the dollar sign after the letter N. The letter is being used as a variable, but the dollar sign makes it a particular type of variable called a string variable. A string is simply a collection of the characters we would normally place between quotes for a PRINT command.

In Program Listing 1, N$ (pronounced en-string) is a string variable which we have declared as a code for the words, "THIS IS A STRING, 1,2,3, TESTING". Each time we ask for N$, that's what we get.

You may not realize it yet, but this is a mighty powerful instruction. It means, for example, that you can print a phrase of up to 250 characters or so just by using the command PRINT N$. Even better, the Level II machine lets you use many string variables.

You can use each letter of the alphabet, a letter and a number (A1$, B3$, and so on), or two letters (AZ$, BD$), as long as the string sign is used to instruct the computer that this is a string variable. If you leave out the string sign, the computer will normally reject any attempt to equate the variable code letter to a string of letters, because a letter with no string sign means that the variable is a number.

The exception occurs at the very start of a program, when you have told the computer that all variables which start with a specified letter will be string variables. This is done by using the DEFSTR (define as string) command. A program starting with 10 DEFSTR A,K,T uses variables such as A, AM, AA, K1, KZ, TZ, TT, and so on without the string sign after them.

Let's look a bit harder at this string thing. Can we really store a long string? Try Program Listing 1 again, but this time make the first line read:

```
10 N$ = "THIS IS A MUCH LONGER STRING WHICH WILL
   NEED MORE MEMORY SPACE THAN THE PREVIOUS ONE"
```

Don't change the remaining lines, but type in the new line 10, ENTER, and run.

The Clear Command

So, you got an error message? Even if you typed everything correctly and didn't get the SN error message, you still get the words OS IN 10. OS means out of string space; was Sinclair wrong? Something else you have to learn about the TRS-80 is you have to let it know in advance how much memory it needs to reserve for strings.

Normally, when you first switch on, the 80 reserves 50 units (bytes) of memory for strings. Each character of a string, and that includes spaces, remember, takes up one byte of memory, so you don't need to have very long strings to total over 50 characters. To reserve more space, use the CLEAR command at the start of a program. Try it by typing in the following line at the beginning of the program:

```
5 CLEAR 200
```

Leave line 10 as it is; ENTER and run. This time there should be no OS error message, because we've reserved enough string space for 200 characters. Now this may seem confusing, because when you are inventing a program you may not know just how much string space you need. That's OK, because you don't have to enter the CLEAR instruction until your program is complete and ready to run, and by that time you should be able to tell how many characters are going to be stored as strings. If you forget, it's no great hassle to type in a line 5 with a CLEAR instruction, followed by a number big enough to store all your characters. Lines are numbered in tens in order to leave room for second thoughts like this.

Why should we have to do this? Well, it's all tied up with the way the computer controls the memory space. We said in the first chapter of this series that it is possible to reserve space at the top of memory for machine-code programs.

This is not the only reserved space in the memory. The memory space just below the machine-code space is reserved for strings. If you haven't used a CLEAR (number) instruction, only 50 bytes of this memory are reserved. Use more than 50 bytes of string and you get the OS warning, because you have run out of reserved space, and that part of memory is in danger of being used for something else.

Why don't we just start every program with CLEAR 2000, reserving plenty of space? Simple: It's wasteful. Reserve too much space in memory and it's like roping off half a parking lot—you're wasting space. Memory is valuable to the computer, so we don't reserve any more than we need, especially when we're entering a long program.

The way computers use strings (called string handling) is one of the points that sets apart the serious computer from the "just-for-fun" machine. It's the big, big improvement of the Level II machine over the Level I, for example.

The little program that we've been running gives you a taste of this. In line 40, the PRINT command asks for a print (on the video screen) of the message we've coded as N$, but also for the message "; ALL WELL." Notice the positions of the quotes and the semicolons? The semicolon immediately after N$ is a command, meaning put in a space and keep printing on the same line. The semicolon inside the quotes is part of the message and it gets printed. There's nothing to show, when you look at the whole message on the video screen, that one group of characters was stored as a string and the other as a PRINT command inside quotes.

Here we should mention the matter of numbers. If a variable letter isn't specified as a string by the dollar sign or the DEFSTR command, then it's a number. We'll find later on that we can define three types of numbers, but for the moment we won't look for complications. We can write a line, such as 20 A = 15, and then throughout the program we can use A instead of having to type 15. If we want to change it, we use another statement, such as 100 A = 16. These statements are called variable assignments, and the equality sign doesn't mean equals when it's used in this way, but rather "takes the value of."

This is very important, as you'll see later, because some statements look odd if you assume that = means "equals." Take a look at the short program in Program Listing 2. N$ is a string variable which we set to be "GREEN BOTTLES" in line 10. The number variable A is set to 10 in line 20. When we get to line 30, we get. . . well, try it for yourself! If we now add a new line:

```
35 A = A - 1 : GOTO 30
```

and RUN again, we see some wild printouts which won't stop until we press the BREAK key.

What happened? We did say that = means "takes the value of." In line 20, A takes the value of 10, so in line 30 you get:

```
10 GREEN BOTTLES, HANGING ON THE WALL
```

(You did get the comma *inside* the quotes, didn't you?) At the new line 35, A takes the value of 10 – 1, which is 9. The colon marks a new instruction on the same line. This saves us from having to make a new line number. The next instruction is GOTO 30—go back and carry out the instruction in line 30 and go on from there. This is the PRINT instruction all over again, so you get:

9 GREEN BOTTLES, HANGING ON THE WALL

The program then automatically steps to the next line, 35 again. This time A starts at 9; the instruction A = A – 1 gives A the new value of 8 and so on. This is called a loop—the program simply goes from instruction 30 to 35, then back to 30 again, and you can't get out of it except by pressing the BREAK key, by another program instruction, or by letting it run out of numbers.

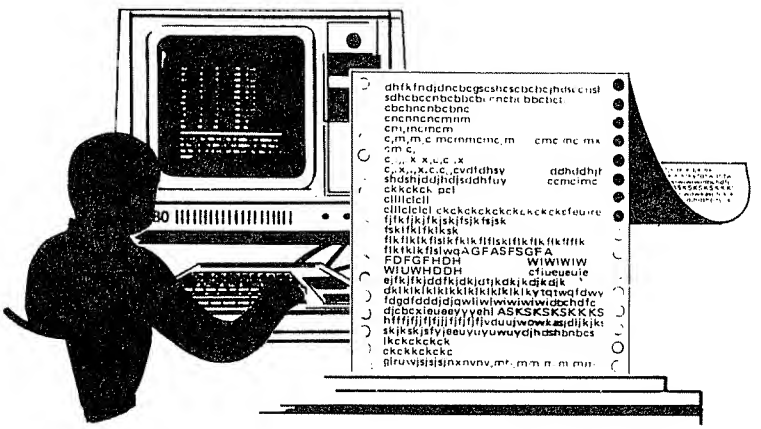
The INPUT Instruction

We need to look now at a more immediate way of entering information into the computer. So far, every string and number we've used has been planned ahead and put into the program from the beginning. The only method we have of changing things is by retyping the program lines (I'll talk about editing them later).

The instruction that saves us a lot of time is called INPUT, and an example of its usage is in Program Listing 3. Type in the program, remembering that the @ sign *must* follow directly after the T of PRINT, no spaces allowed, and the number of the PRINT@ position must be followed by a comma and then the first set of quotes. If you run the program, right away the screen clears, and the words

WHAT IS YOUR NAME?

appear. On the next line a question mark appears, and the program stops, waiting. It's waiting for you to put in your name, or any other name. You can take your time about typing a name, because the computer waits until you hit ENTER. When you do hit ENTER, your name appears with that famous phrase after it. You can enter any name or any gibberish at the INPUT step. It will accept numbers, or mixed names and numbers like CONVICT 99, or anything else you put in. They will get printed just as if they had been placed between quotes in a PRINT command.



This is more useful, because it lets you write programs that look a bit more friendly, for a start. The TRS-80 goes further with its input command than some others, in fact, and lets you use INPUT like a PRINT statement, so you can write a line such as:

```
20 INPUT "WHAT IS YOUR NAME"; N$ : CLS
```

to replace line 20 and 30 in Program Listing 3. Do I hear an objection? It's true that when you use INPUT to print like this, you can't place the printing where you want it, because you can't have INPUT TAB or INPUT@. Try this for line 20:

```
20 PRINT@22,;:INPUT "WHAT IS YOUR NAME";N$:CLS
```

Watch the sequence of delimiter markings in this one—after the 22 we have comma, semicolon, and then colon marks. Notice we don't use a question mark after NAME, but you'll see one when the program runs, because it forms part of the reply to the INPUT command.

Suppose you try to use N instead of N$ after INPUT? You can't do it, unless what you enter is simply a number. If you specify a string variable, you can INPUT what you like, up to 255 characters; however, if you specify a number, then you must enter a number—no letters permitted.

Using INPUT statements to make a sort of conversation is illustrated in Program Listing 4. In line 20, your name is assigned to N$ by the INPUT statement, and line 30 makes a friendly comment.

At line 40, the INPUT asks for age, and at line 50 for this year. The grand finale is in line 60, when the printout on the video screen gives the name and year of birth. How? Since it has the present year, represented by variable Y, and your age, variable A, it only has to subtract A from Y to get your year of birth—unless you lied about your age! Simple—but it looks like magic to anyone who hasn't seen your TRS-80 in action before.

CLOAD and Friends

CLOAD is one of the instructions we use many times on the TRS-80. It means Cassette Load, and it's the instruction that lets you use these programs on cassette.

The freedom that cassette loading and saving gives you is immense. Without cassettes, each program you use is lost whenever you type a new program or switch off. By saving your programs on cassette, you can enter them at any time.

In addition, cassettes give you a chance to run programs which might take many hours to enter from the keyboard or which most of us could never devise even if we were locked in a padded cell for a year.

Okay, let's go over cassette loading in detail. If you bought a complete TRS-80 outfit, you'll have the CTR-80 recorder; a used TRS-80 might come with a CTR-41. I use a fairly high-grade recorder which has a better-than-

normal frequency response (which means it records and plays high notes better). This is advantageous because the recording and replaying of data and programs uses high notes, and you can't load properly unless these notes are loud and clear. For example, if the tone control of a cassette recorder is set to reduce high notes, it just won't load programs.

Whatever cassette recorder you use must have a microphone input socket, automatic recording level adjustment, an earpiece output socket, and a motor control. The motor control takes a small (2.5 mm) jack plug, the smallest plug on the TRS-80 cable, which comes from the cassette outlet on the TRS-80 keyboard.

The signals out of the cassette recorder come from the earpiece socket, a 3.5 mm one, which is linked by pushing in the black plug at the end of the TRS-80 cassette cable. For loading cassettes, you don't need the microphone plug (the grey one). Check them all again.

If you are not using the CTR-41 or CTR-80 recorders, then whatever you use must have the same plug-in arrangements, particularly the motor control, because the recorder motor is controlled by the computer. If you bought only the TRS-80 keyboard and are using your own cassette recorder (or reel-to-reel recorder), then you will have to make or buy adapter leads, or do without motor control.

If you are using the CTR-80 from Radio Shack, then the recorder will run fast forward or fast reverse even when the computer has stopped the motor from running in the play or record/play settings. If you are using the CTR-41 or any other cassette recorder, you won't have this rather useful facility. For a lot of program work it won't matter, but if you would like to go from one place on the tape to another, then there are various fixes. A few user-group magazines will show you how to cut tracks inside the recorder to do this.

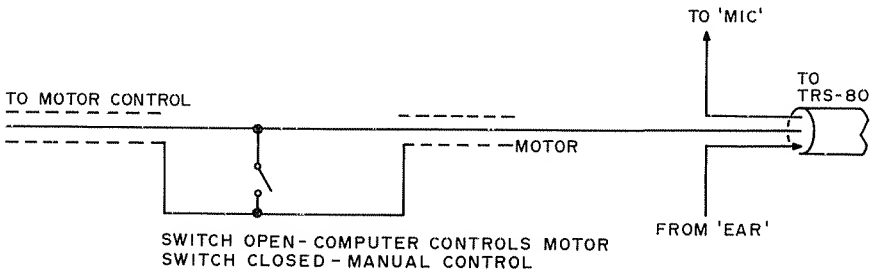


Figure 1

My own fix consists of an adapter box and a small switch which allows either normal or computer control of the recorder motor. With this addition,

you can also use manual control with the cassette recorder switched to play, which is useful for finding a short gap between the programs on the tape. If you start a playback in the wrong place on the tape, it won't load correctly and the program won't run. If hardware doesn't interest you, the easy solution is to type:

```
10000 OUT 255,4: GOTO 10000
```

ENTER this and RUN, and the motor will stay switched on by the computer until you press BREAK.

Loading the Program

At this point we've sorted out our recorder, everything's plugged in, and we're ready to go. Next, we need a BASIC program on a cassette. My TRS-80 came with Radio Shack's Blackjack program, and it's likely that yours did too. If not, then you'll need some software.

Pop the cassette into the recorder, with the program you want to load, so that the label of the wanted program is uppermost. Rewind the tape completely—the CTR-80 will make a moaning noise when the rewind is complete.

Set the volume control of the recorder to halfway between its maximum and its minimum settings. Make sure that the tone control, if you have one, is set to give maximum treble.

Now we're ready. Type CLOAD on the TRS-80 keyboard, press play on the recorder and press ENTER on the keyboard. You should hear the motor of the recorder start to hum. If the motor starts when you press play, there's a fault in the motor circuit somewhere. The motor-control jack may not be plugged fully in. If the motor doesn't start at all, then perhaps there are no batteries or the power line isn't plugged in. (Editor's note: The motor control relay is also notorious for "sticking." To help correct this problem, see "Build a Snooper/Snubber" in the HARDWARE section of this book.) These are what we call hardware problems.

Another possible hang-up could be a software one. Are you sure that you typed CLOAD? Keybounce, which may have given you CCLOAD or CLLOAD won't be accepted by the computer, and it will snap back with an SN error when you hit ENTER.

By now, if all has gone well, the cassette should be running. Unless you have connected the loudspeaker of the cassette recorder so that you can listen to the tape as it plays, you won't know when the action actually starts, until you see things happening in the top right-hand corner of the monitor.

Two asterisks appear once the program starts loading, one steady, the other flashing slowly. One asterisk flashes at the rate of loading program lines, on for one line, off for the next. If you're loading a short program with short program lines, the rate of flashing will be rapid, and it won't be long

before a click comes from inside the keyboard unit, the cassette recorder motor stops, and READY appears on the video screen.

If all this happens, you have achieved a successful load first time, and you qualify for the Fort Worth Perfboard Medal of Honor.

It's much more likely, first time around, that things won't run quite so smoothly. There are two extremes to the problem. One is that no asterisks appear at all. This could simply be due to a tape which starts only after a long leader, which is why it is so useful to have a loudspeaker tone; but if there is no trace of the asterisks after a minute, there is replay volume trouble. Despite what the manual may say, this indicates that the replay volume is either much too low or much too high. If, on the other hand, you get two asterisks, but the right-hand one isn't flashing, then it's a dollar to a cent that the replay volume is just a little bit too high.

Cassette Control

If you have either of these problems, you'll soon find that you have another one as well. The cassette recorder motor keeps humming away happily until it comes to the end of the cassette or until you do something about it. It certainly won't stop at the end of the program load, because the stop instruction was never loaded into the computer. You can waste a lot of time just waiting for a cassette to load, so keep a careful eye on these asterisks. If they aren't blinking properly, then press the RESET button at the back of the computer, rewind the cassette, press CLEAR to remove the old instructions and asterisks, and start again with a different volume control setting.

Don't give up if you overshoot and go far too high or far too low. When I bought my first TRS-80, I spent the better part of an afternoon trying to achieve a good load. Since finding the correct settings, it has never at any time failed to load a good cassette. You don't need to use expensive chrome-dioxide tape material, just reasonably good quality audio tape, like Agfa or TDK. It's definitely an advantage to use tape sold in short lengths for computer work, but the C60 length is very useful when you're developing a program with several versions.

You may find that you have a tape which simply won't load under any conditions. The odds are that it's a tape intended for a Level I machine. Once again, if you have the sound wired on your cassette recorder, you can check this, because the Level I tapes have a lower pitched note and sound quite different.

Once you have found the correct setting for the Radio Shack Blackjack tape, or whatever you use for trials, try to find the limits of the volume control settings. It takes a lot of patience, but it's worth your while on the older models (later models are more tolerant of volume control settings). Load the cassette at different settings, checking only for a few seconds that the

asterisks are flashing correctly, then use RESET to stop the tape and rewind. You should end up with two marks on the volume control, one at the lowest position at which a tape will load, one at the highest.

Set the volume control for normal operation midway between your marks. If you then find a tape doesn't load correctly at this midway setting, try it at each of the extreme settings. If it won't load on any of your marked range, reject it. One final check: Make sure it is a BASIC tape and not a machine-code (SYSTEM) tape, which requires quite a different technique.

When you have loaded your BASIC tape, type LIST and hit ENTER. The program will now list, unless it's one which has been specifically coded to prevent copying. As the program lists, it will scroll up the screen fairly rapidly.

You can stop the scrolling any time by hitting SHIFT and @ at the same time, so I usually keep one finger on the shift key and another on the @ key to stop and start the scrolling. Any other key can be pressed to restart scrolling.

I didn't say what scrolling means? When you've seen it, you'll know—it's the way the listed lines appear at the bottom of the screen, seeming to push previously listed lines off the top of the screen.

What you should be looking for in the listed program is corruption—gibberish lines, sometimes with no numbers or numbers out of order. Trouble is, until you get to know a bit more about programming, you don't really know a strange line from a perfectly good one! The real test, of course, is to run the program. If it operates perfectly, then there is nothing wrong with your cassette recorder volume control setting, and you can look forward to a long, active computing life.

Just keep your head clean. I mean, of course, the record/play head of the cassette recorder. Get a pack of cleaning fluid (isopropyl alcohol) and cleaning pads and use them as the instructions indicate: every three months or so, depending on how much tape you use. Don't be too generous with the fluid, as it can sometimes swell the plastic bearings inside cassette recorder motors.

Loading System Tapes

While we're on the subject of this cassette loading caper, we might as well look at how machine-code tapes are loaded. A machine-code, or system tape usually comes with a bit more information than a BASIC program tape.

For one thing, you'll need an answer to the MEMORY SIZE? question which appears when you switch on. This is a number, such as 32000, that reserves some memory. The system tape, or the instruction sheet which comes with it, should have the correct number printed on it. If the program is one which doesn't need reserved memory or which reserves its own, the instructions will say so.

Hit ENTER and the usual Radio Shack message comes up, with READY. Next, type SYSTEM, and hit ENTER again. This time, you'll get an asterisk and a query at the left-hand side of the video display. That's 80 language for, "What's the code name for the tape?"

The code name will have up to six letters and must be typed. For example, the Radio Shack fix tape for keybounce has the code name KBFIX. When you've typed the name, prepare the cassette recorder to replay, press the play key, and hit ENTER. The cassette recorder motor will start, and, if all is well, you should see the usual asterisks to indicate that you are loading your first machine-code program. The rate of flashing is usually a lot slower than it is for a BASIC program, so don't worry if the load stops after only a few slow flashes.

Now what can go wrong? Well for one thing, the code name which you typed may not be the code name for the first program on the tape. If it isn't, the left-hand asterisk will be replaced by a letter. If that letter is C, then you have trouble, and you'll have to try again with a different volume setting. Hit the RESET switch to stop the action, rewind the tape, clear the screen, and start again by typing SYSTEM. You don't have to switch off and answer the MEMORY SIZE? question again.

Next question: Having loaded it, how do you run it? When the tape has finished loading, the recorder motor cuts out, the asterisk stops flashing, and another asterisk and query appear under the first one on the left-hand side of the monitor. Type a slash (/) and then the entry number of the machine-code program.

Machine-code programs are not as simple as BASIC programs in this respect: You have to instruct the computer where to start working. The entry number should, once again, be noted either on the cassette or on the instruction leaflet. It may be the same as the number you used to answer the MEMORY SIZE? question. When you've typed the slash and the number, hit ENTER and your machine-code program will run.

Suppose you quit using one machine-code program and want to start using another one which needs more memory roped off? You don't have to switch off to do this. Just type SYSTEM, hit ENTER and when the asterisk and query appear, type slash and 0 and hit ENTER again. The MEMORY SIZE question will appear again. You'll lose any BASIC programs you had in store, though, so if you have mixed BASIC and machine-code, make sure that you have the BASIC program on tape.

A few machine-code programs are "self-locating." Once you have loaded them in by typing their code names and entering, the second step is just to type the slash and hit ENTER. Whatever type of machine-code program you may be using, don't forget the slash. Otherwise, you'll find that when you hit ENTER, the cassette motor starts running again, trying to enter another program, and you'll have to recover control by using the RESET

button. You'll probably lose the program which was loaded, but you can start again.

Recording Programs

We've left until the end the matter of recording BASIC programs of your own. You'll want to record your own programs, of course, to remind yourself how good you are. You'll also want to make back-up copies of software you've bought, just in case anything should happen.

How do you record a program? The first step is to prepare a blank cassette. Don't think you can re-record an old tape in the same cheerful way you may be used to doing with audio cassettes. You might get away with it, but odds are you won't, and your recording will be corrupted. If you want to re-record a tape, wipe it completely with a bulk eraser. If the program you want to record has taken you a long time to run correctly, you won't want to trust it to anything but a length of good quality fresh tape.

Reel the cassette back to the start and take a look at it. If there's a leader, a piece of clear or colored plastic tape at the beginning, advance the tape a bit until the grey magnetic coating is visible. I usually run each tape for a count of five on the tape-footage counter. Don't touch the tape; it will leave a greasy mark which can cause loading problems later. Place the tape in the machine again, note the counter setting, and press the record and play keys. A few cassette recorders use one single record key, but most use the safer system of needing two keys for recording.

The volume control setting doesn't matter, because recording volume is automatically controlled, unlike replay. Now type CSAVE and a quote mark, then a letter and another quote. If you choose "A" as the letter, this will appear on the screen as CSAVE "A".

If you don't use the code letter the computer will reject your attempt to record, but only after it has already recorded a signal on some of the tape, which you won't be able to use again unless you can erase it thoroughly.

When you're satisfied that all is well, hit ENTER, and the program should start to record. There are no flashing asterisks to remind you this time, just the quiet hum of the motor of the cassette recorder until it clicks off at the end of the recording. The click, incidentally, comes from the relay inside the TRS-80. At the end of the CSAVE, READY appears on the screen.

At this point, don't start shouting eureka and running around. You don't know yet that you have a good recording. Rewind the tape, type CLOAD?"A" (or whatever letter you used), and press play on the recorder. Then check again that the query mark has been typed after CLOAD, hit ENTER, and wait. The program will play back, with the usual flashing asterisks, but this time the replayed program is being compared byte by byte with the program which is still in the memory of the computer.

If they aren't identical, the message "BAD" will be displayed. You have then to sort out whether the tape copy is faulty or you need a different volume control setting for this program. Only when you've CSAVEd and CLOADed with no error messages can you be sure that you have a good copy of your program. Cautious people always make two recordings, checking one with CLOAD? People like me who shed blood, sweat, and tears to create a program always make three copies.

Be very careful that when you use the CLOAD? command, you don't leave out the query mark. If you do, the program on tape will load, replacing the program that was in the computer. If the recording was good, this won't matter, but if the recording was bad, you have lost the good original and have a bad copy, and that just isn't fair trading.

The CSAVE Instruction

Very little ever seems to go wrong with a CSAVE instruction, but there are a few points you will need to remember. One is that the computer can only control the motor of the cassette recorder; it has no control over the rest of the recorder.

If, for example, you use CSAVE but forget to press the record and play keys of the recorder, or press only one of them, then the computer will push out the recording just the same, with no warnings and no recording made. It might be useful to arrange it so that you got an error message, but this would need more connections between the computer and the recorder and would make the recorder a non-standard item.

You should always use CLOAD? after a CSAVE, so you can check that you really did record that program. A much worse fault is to type CLOAD and run with the record and play keys down. This way you load no program, and you wipe out anything which was on the tape!

When you CSAVE a program, you have to use a letter or a couple of letters or letter/number—it's like choosing a name for a variable. If you don't do as we've said, the CSAVE will not run, an SN error will be displayed, and the tape will be corrupted.

The label is called a file name, and it's important to the recording. It's used when you CLOAD the program, and it's particularly useful when you have several short programs packed together on a piece of tape. Suppose you have three programs on the start of a C15 cassette, and they have been labelled "A", "B", and "C" at the time they were CSAVEd.

When you CLOAD, you can type CLOAD?"B" and hit ENTER, and start running the cassette from the start. When the first program starts to replay, the left-hand asterisk will be replaced by the letter A to show you that this is the file name (the first letter if there's more than one) of the program which is being read. The other asterisk will flash normally. When the

program which you have requested comes on line, it will load in the usual way, with one steady asterisk and one flashing one, then the recorder will switch off.

Normally, when I keep several programs on one cassette, I leave plenty of space between and use the tape counter to find each one, but I find this label-search very useful for my backup cassette, which is a C60 with all my most valuable programs stored tightly together. Since I use this only when a valuable program has been wiped or corrupted (and I'm resting the *other* backup cassette), it doesn't matter if it takes twenty minutes to find the program.

One last point—always start a replay either at the start of a cassette or at a point where you know there's no program recorded. If you start running where there's a program recorded, the load will be faulty, and the computer can lose control of the motor. You'll end up having to use the RESET button and rewinding the tape.

Please note: These listings are not formatted. Enter them normally.

Program Listing 1

```
5 REM FIG.2.1 INTO 80'S
10 N$="THIS IS A STRING , 1,2,3, TESTING"
20 PRINT N$
30 PRINT
40 PRINT N$;"; ALL WELL"
```

Program Listing 2

```
5 REM FIG 2.2 INTO 80'S
10 N$="GREEN BOTTLES"
20 A=10
30 PRINT A;;N$;" ,HANGING ON A WALL"
40 END
```

Program Listing 3

```
5 REM FIG.2.3 INTO 80'S
10 CLS
20 PRINT@23,"WHAT IS YOUR NAME?"
30 INPUT N$: CLS
40 PRINT@17,N$;" -THIS IS YOUR LIFE!!"
50 END
```

Program Listing 4

```
5 REM FIG.2.4 INTO 80'S
10 CLS
20 INPUT "WHAT IS YOUR NAME, PLEASE";N$
30 PRINT:PRINT N$;" I LIKE THE SOUND OF THAT"
40 INPUT "TELL ME PLEASE WHAT AGE YOU WILL BE THIS YEAR
 (IN WHOLE YEARS)";A
50 INPUT"AND NOW WHAT YEAR THIS IS";Y
60 PRINT:PRINT"SO, ";N$;" ,YOU WERE BORN IN ";Y-A
70 END
```

Into the 80s Part III

by Ian R. Sinclair

Now that we've been over the methods of CSAVE and CLOAD, we can take steps which lead to longer programs. I am going to explain the programming methods which you'll find in longer programs and show some short examples which you can use in programs of your own.

Are you ready for the Force? The instructions we're going to look at in this part are among the most powerful instructions in BASIC, and your TRS-80 has one of the most complete BASICs I know.

The IF-THEN Statement

The IF-THEN-ELSE instruction allows a computer to make a comparison and a decision. The comparison will be between two quantities, strings, or numbers, and the decision will be about what to do next. The best way to show how this works is with an example (Program Listing 1). Let's go through it carefully.

Lines 10 through 60 print out the rules of a very simple game. A lot of improvements can be made, and we will need to make them if we want the game to be interesting, but for the moment, let's take just one step at a time.

The new parts of the program start at lines 70 through 90. The program prints the word LION and waits for your reply to be typed and entered in line 80. The reply which you type becomes the variable N$, which can now be compared with the correct answer, which is the word PRIDE. Line 90 does just this: If you typed PRIDE, correctly spelled, then N$ = "PRIDE", and the program will print the words WELL DONE and end.

If you typed anything but PRIDE, the rest of line 90 is ignored, and the program shifts to line 100 to tell you that your answer is wrong. You are then asked to try again, and the program returns to line 70 by using the command GOTO 70. Try it, giving a correct answer on one run and an incorrect answer on the next run, so you can see how the computer treats these different cases.

Meanwhile, what about ELSE, which only a few computers feature in their BASIC? The BASIC statement in line 90 used only IF-THEN. *If* N$ = "PRIDE", *then* the program goes on to complete the other instructions in line 90. *If* N$ is not "PRIDE", *then* the rest of line 90 is ignored and the next line executed is line 100. That last section of line 90 is rather important, incidentally. If you omit the :GOTO 120, when you answer PRIDE the computer would print:

```
WELL DONE
WRONG, I'M AFRAID—TRY AGAIN
LION ?
```

A correct answer should stop this simple program, and only an incorrect answer should permit the entry of another answer. You have to remember when you write a program that unless you command it otherwise, the program will always step from one line to the next in numerical order.

The ELSE Command

That big, big BASIC of the TRS-80, however, lets you write lines 90 through 120 in a much shorter form, which is shown in Program Listing 2. This can now be the last line in the program. Type in DELETE 100.120, hit ENTER and then type in your new line 90. Try it; this time, if N$ is not "PRIDE", the rest of the line is ignored only as far as ELSE, then the section after ELSE is carried out. Using IF-THEN-ELSE in this way can save a number of lines in your program.

Computer Comparisons

In addition to the use of IF-THEN-ELSE, another innovation is the use of the equality sign in the expression IF N$ = "PRIDE". This is not quite the same use of the equality sign that we've used until now. When we have a command like IF N$ = "PRIDE", the computer compares the two stored strings, N$ and PRIDE, letter by letter, to determine whether they are identical. If one string has a space or a comma or a period and the other hasn't, then they're not identical. We'll later look at ways around that problem.

The equality sign comparison isn't the only one which can be made. We can also write IF N$ > "PRIDE" or IF N$ < "PRIDE", though these statements would not be used in this game. The > sign means greater than, and when it's applied to a string it means that the word used for N$ would come later in an alphabetical index than the word PRIDE. For example, if "ROAR" > N$, then it comes later in a list than PRIDE, because R follows P in the order of the alphabet. "PRUDENT" > N$, because it also comes later, because U comes after I in the alphabet, even though both words start with PR. The < sign means less than, and works exactly in reverse. To complete the story, we can combine these symbols as shown in Table 1.

Clearing Methods

Since we're writing programs of twelve lines and more, we need to be able to clear one program (after using CSAVE to preserve it) in order to start all over again with another program. Type NEW and hit ENTER —it's that easy. This doesn't actually erase the program the way you can erase a tape, it erases only the instructions inside the computer which act as a signpost to the

start of the program. Your old program is completely wiped out when you enter a new one of the same length or longer or when you switch the computer off and on again later.

Some owners of other computers would give both ears and a tail for the TRS-80's edit facilities. We're not going to cover all of the editing methods at once, but it's time you met the main one. With your program set up, type EDIT 70 and hit ENTER. This will result in the number 70 being displayed on the screen with a cursor (dash mark) beside it. Press the space bar and release it, and the cursor moves right. Press again, and the first letter of PRINT appears. Another press and the second letter appears. Looks as if you're typing all these letters with the space bar, doesn't it? The backshift arrow (←) allows you to go back until just the number shows; the space bar allows you to go forward to show more of the instruction.

Space bar your way to the end of the line and then backspace until the last quotation marks disappear but you can still see the entire word LION. Press the letter I on the keyboard, but don't hit the ENTER key. Backspace until the L of LION disappears, leaving only the first quotation marks visible. Now type the word WHALE and hit ENTER. The line should read:

```
70 PRINT "WHALE"
```

The new word has been inserted (I for INSERT) between quotation marks. In this example, we first had to delete by backspacing after the I had been pressed, but it's also possible to add letters, spaces, or whole words into a line by using the I key and then typing in the new material. You can alter a line as much and as often as you like in this manner, but if you interrupt a program to alter a line, you will have to reRUN the program from the beginning.

Now that we've changed line 70, we need also to change line 90. Type EDIT 90 and hit ENTER. Use the space bar to step along to the E of PRIDE, then press the I key. Step back, using the back arrow, until the P of PRIDE has disappeared, then type in SCHOOL and hit ENTER. Line 90 should now have "SCHOOL" in the place of "PRIDE", and the program makes sense again.

Increase the Beasts

One of the problems of our program in Program Listing 1 is that it's limited, to say the least; it's not the sort of thing that's likely to hold your interest on a long rainy afternoon. Perhaps we can use a new instruction to pep things up a bit, starting with a method to add more animals.

Look now at Program Listing 3. There's a new instruction in line 70, READ Q$,A$. The READ instruction tells the computer to look for data, and the data must always be labeled by starting with the word DATA. There's no comma after DATA, but there must be a comma after each word in the list except for the last one. Because we're asking the computer to read

string variables from this list, it is a good idea to enclose each word in the list within quotation marks. Where the comma after each word is *not* inside quotation marks, it indicates to the computer where each word ends.

In line 70 the computer assigns values to the string variables Q$ and A$. First time around, it makes Q$ identical to LION and A$ identical to PRIDE. To do this, the computer simply makes the first string variable, which is Q$, equal to the first word read from the data line, and the second string variable, A$, equal to the second word read from the line. We can have a line 70 which looks like this:

```
70 READ Q1$,A1$,Q2$,A2$,Q3$,A3$
```

This would have read three sets of question and answer words, or we could have read all six sets in one operation.

As it is, we chose to read just one question and one answer in line 70, and in line 80 we print the question word. Since Q$ is assigned to LION in line 70, that's what comes up on the video screen. We don't ask for the answer word (A$) to be printed, so it isn't. At line 90 you're asked to input your answer, and line 100 then compares your answer with one, PRIDE, which has been taken from the list.

We've made a few changes in line 100, also. If your guess is correct it is announced on the video screen, and the instruction GOTO70 tells the computer to read another pair of words. That's what makes this READ . . . DATA pair of instructions so useful; each READ is a new one, with new information coming in from the data line or lines. This time, Q$ is set equal to WHALE, and A$ is set equal to SCHOOL. See why we call these quantities variables? We vary what they are set to each time, instead of leaving them set for all time.

Looks a bit more interesting now, doesn't it? You can use as much data as your computer has space for (and your typing fingers will really ache before you fill up the 16K TRS-80 with data). Your TRS-80 won't let you type more than a total of 255 characters on a line, so if your words (or numbers, of course, if you use number data) need another line, then you just start another line.

There are rules about this, as you might expect. The last word in a line must not have a comma following it, and the next line must start with a line number which is greater than the line number of the previous line. After the line number, you must type in the word DATA, then the first data word for the line, a comma, the next data word, and so on.

The computer always reads the data in order, starting with the lowest numbered line. There is no simple command which will fetch word number six, for example, although such a command would be very useful to have. Later we'll see how we can get around this limitation.

We still don't have a really satisfactory program yet. For one thing,

there's no end to the program. It simply reads data until the last word has been read, and then you get an error message—OD (out of data). If you can't answer one of the questions, the program simply sticks, going back to line 80 from line 100 until you answer correctly or switch off in disgust.

Change Your Game

We need a few changes. First, we need to be able to stop the program when all six sets of words have been used. Secondly, we need to be able to limit the number of wrong answers so that the program doesn't stick. Finally, it would be useful to keep some sort of score.

You may not realize it, but you know one method by which to make these changes. The obvious method is to use counting variables which start at zero or unity and are increased by one (incremented) at each loop of the program.

Start by counting the number of times a set of questions and answers is read from the lines. Do we need to count this? Counting is one way of solving the problem, but there's another one: Put in a final pair of data items, and make the computer reject them. There's no animal called Z, and it doesn't hunt in Zs, so we can add Z,Z to the end of the line. We don't want to print Z, so we'll intercept this data, called a terminator, between reading in line 70 and printing in line 80:

```
75 IF Q$ = "Z" THEN END
110 DATA "LION","PRIDE",. . . "GEESE","GAGGLE","Z","Z"
```

We have to put both Zs in the line, because the READ statement in line 70 always reads two strings. If there's only one, we'll get that OD error message again. This is a much more satisfactory way of terminating a read than by counting the number of sets of reads, because it lets us add to the data easily, by inserting more data between the gaggle and the Z; if we had used a count, we should also have to change the count number.

We now have the problem of the program looping around line 80 through 100 and back when you can't answer. Let's allow three tries only, and if all are wrong, we print the correct answer and fetch the next pair of words.

How do we do that?

First of all, we must select a letter to represent the number of tries; T looks useful, as it will remind us of t for try. A letter which reminds you of what you are trying to do makes life a lot easier when you are designing and redesigning the program, or when someone else is trying to understand it. We want to allow one attempt whenever a pair of names is read, so we need to make T take the value of unity each time data is read. Edit line 70 to:

```
70 READ Q$, A$: T = 1
```

and T will be correctly set at each read step.

Now we want to add one to T each time you answer incorrectly. We can

do that by altering line 100 (which is lengthening every time we alter this program) to read:

```
100 IF N$ = Q$ THEN PRINT "WELL DONE":
    GOTO 70 : ELSE PRINT
    "WRONG, I'M AFRAID - TRY AGAIN":
    T = T + 1 :GOTO 80
```

After our third attempt, T will have a value of four. We now need to arrange for this to cause the program to break out of its loop. If T is less than four, line 80 should print Q$ and ask for an input reply. If T is equal to four, we want to print the correct answer and start with another pair of words. It looks like a convincing case for an IF-THEN statement. Suppose we make line 80 read:

```
80 PRINT Q$ : IF T = 4 THEN PRINT
   "ANSWER IS";A$:PRINT:GOTO 70
```

If you've had three attempts, the answer is printed and a new animal question is asked. In line 70 T is again set to 1, so the next time the program goes to line 80 the new piece of program is ignored again. We've printed the words and the variable A$ in the new section of line 80 using a semicolon to keep the video display running on the same line. (We could have used T = 0 in line 70 and IF T = 3 in line 80.)

The next item on the list is a way of keeping score. To be fair, we need to keep a tally of the number of total attempts and the number of successful attempts. Each time we've been successful, we've printed WELL DONE, so we could make a count of the successful attempts there. Each time we answer, we input something on line 90, so the total number of attempts could be counted there.

Let's use the variable A for the number of attempts. We have to start at zero, so A must be set to zero early in the program. Line 10 is fairly empty, and we can add, after a colon, A = 0. To count the attempts, line 90 needs another addition: A = A + 1 so that A is increased by one each time you answer.

If we use S to count successes, we can set S = 0 in line 10 and increment it just after the statement PRINT "WELL DONE" in line 100:

```
100 IF N$ = Q$ THEN PRINT "WELL DONE":
    S = S + 1:GOTO 70: ELSE
    PRINT "WRONG,I'M AFRAID - TRY AGAIN":
    T = T + 1:GOTO 80
```

Finally, having counted attempts and successes, we better make some use of them. When the last pair of items (the terminators) has been read, we can print the scores instead of just finishing the program. This is done by adding line 75:

```
75 IF Q$ = "Z" THEN 120
```

and adding line 120, which prints the score.

In case you're getting a bit lost with all these changes, Program Listing 4 shows what the program now looks like. The program which started as a very simple game is now more advanced and does its own scoring as well.

Add Excitement with FOR-NEXT

The game will be much improved if we can arrange the program so that the computer can pick any animal at random and surprise you. We can't tackle that until we learn two other instructions.

The first is a really powerful one called the FOR-NEXT loop. Its purpose is to allow you to count the number of times an operation is carried out. For example, if we type in the instructions:

```
200 FOR N = 1 TO 6
210 READ S$
220 NEXT
```

a loop will be set up to read six items from a line somewhere else in the program. The first time the computer comes to line 200, it sets N at 1 and then in line 210 reads the first item, assigning it to S$.

There being no instructions about what to do with the item in this example, the computer goes to the next line—NEXT. NEXT means go back to the FOR instruction and make N one step greater. The size of the step, unless you instruct it otherwise, is 1. The next time round N is set to 2, and in line 210, the second item is read.

Once again we go to line 220, and the NEXT instruction compares the value of N (now two) with the limit we set (which was six) and returns the program to line 200. This loop repeats until the NEXT instruction makes N = 7. This stops any return to line 200, so that the program goes on to the next line.

The example we've used is a fairly simple one, with very few instructions between the FOR and the NEXT. We could, in fact, write such a short piece of program on one line:

```
200 FOR N = 1 TO 6 : READ S$ : NEXT
```

and we don't have to worry about having to set up a comparison like:

```
210 IF N< = 6 THEN 200 ELSE 220
```

The word powerful, when it's applied to an instruction, means it does a lot of program work without needing much typing. The amount of work it does can be judged from the time it takes. As an example, and so that you can see the FOR-NEXT loop doing something, try the program in Program Listing 5. Use a digital watch to measure the time between pressing ENTER on this one liner and getting the READY signal back, and watch the line

printing out. FOR-NEXT loops are often used deliberately in programs to create a time delay, such as to give you a definite time period in which to answer a question before moving on to the next one.

BASIC Information

There's a routine built into the BASIC language which picks numbers randomly for any number limits you like to use. The command is RND, and what makes it so useful (not just for games, incidentally) is that it can be followed by a whole number (an integer) in parentheses. The result will be an integer picked at random which lies between one and the number you used in the parentheses. For example, RND(6) should cause the computer to come up with a random whole number between one and six.

We have a data list of six items and can produce a random number between one and six. It would be useful if that random number could be used to select the corresponding item of data. For example, if RND(6) came up two, the second item from the list would then be selected, and so on.

There's no such instruction in BASIC so we have to look for ways around this problem. Suppose the random number came up three. Could we perhaps read the data list three times and use the last item only? We could indeed, and that's what the first sample FOR-NEXT program did.

Take a look now at Program Listing 6. There's a new line in the old program, line 65. At the start of line 65, T, the number of times you've tried, is set at 1. We had to shift it because our new program is going to read data in several times before it actually prints an animal name, and we don't need T set more than once each time. The next instruction in line 65 is Y = RND(6), which picks a number between one and six and allocates it to the variable Y. We can now use a FOR-NEXT loop, with the counting variable N counting from one to Y. You don't know yet what that number Y is, as it's going to be set and used by the computer itself.

What happens on each loop? At a value of N set at one, the program moves to line 70 and reads the first two items (LION, PRIDE) on the list. There are no other instructions, so the NEXT command causes N to advance to two and the next pair of items is read. Reading the next pair of items in this way automatically causes the previous values to be wiped out, just as recording a new item on a music cassette wipes out the one on it before. The value of the variable N will be increased by one on each run around this loop, until it equals Y, the random number. Suppose Y happens to be four. Then the fourth set of words is read and the loop stops with Q$ and A$ storing the fourth set of words, SHEEP and FLOCK.

The FOR-NEXT loop has stopped and the program moves to line 80, carrying out one very important instruction on the way. RESTORE causes the data selector to go back to the start of the data. Without RESTORE, the next

time we look for a word the data would be counted from where we left off first time, which doesn't leave much room for choice, since only a herd of cows and a gaggle of geese follow the flock of sheep. RESTORE sets everything back so that the next random number starts the search from the beginning of the data again.

We've now arrived at line 80, and the question word is printed as before. The rest of the program is also unchanged, so that if you answer correctly or have three unsuccessful tries, the program returns—or does it? You need extra eyes in this business. If we want the next word to come up, we need a new random number, else the program will go back to its old way of taking the next pair of data words. Instead of GOTO 70 in line 80 and 100, we want GOTO 65, and that should set things right.

The game's getting more interesting now, and it would be useful to have more items on the list, because with only six sets of items it's not much of a game. Our changes have made the Z,Z terminator unnecessary. Because we're picking at random from six, there's no chance that Z will ever be picked, so we can remove these letters from line 110. We can also remove line 75.

How do we go about ending the game and reading the score? It would be useful to see the score any time we want and opt to continue or end.

For a simple game like this, let's view the score after each set of five answers.

We need to count a set of five items printed and then show the score. We will set up another variable (J) to act as a counter and increment each time a question is printed. We want a way of telling when J is 5, 10, 15, or any other multiple of 5. We could have lines like:

```
200 IF J = 5 THEN.....
210 IF J = 10 THEN.....
220 IF J = 15 THEN.....
```

but that's a waste of time and memory. A much easier trick is to make use of yet another feature of that big BASIC in the TRS-80, the INT command. INT means rounding off a number by removing the fractional part. INT(6.25) is 6, INT(2.14) is 2, and so on. The way we're going to use INT is in a decision step:

```
IF INT (J/5) = J/5 THEN.....
```

The easiest way to understand how this works is to imagine taking values from one upwards. If J is 1, then J/5 is 0.2, and INT(J/5) is zero. J/5 certainly isn't equal to INT(J/5). For J = 2,3,4 we get the same effect; the INT value is zero, but for J = 5, when 5/5 = 1, and INT(5/5) also equals 1, the test succeeds.

At J = 6, J/5 = 1.2, and INT(5/5) = 1, and the two are unequal again

until J = 10, when both J/5 and INT(J/5) are equal to 2. This test therefore allows us to detect each set of five steps of J.

If J/5 = INT(J/5) we want a score. We don't want the score to come up too quickly, so we'll introduce a time delay between each test, which will also delay the appearance of the score. To do this we can use:

```
FOR Z = 1 to 500: NEXT
```

Z doesn't mean anything to the program; it's just a variable which we're using for a time delay.

How do we use the test IF J/5 = INT(J/5)? If the test fails, the ELSE at the end of the line directs the program to find another item. This will happen on the first four runs. When the test succeeds, and J/5 = INT(J/5), we've reached the fifth (or tenth, fifteenth, twentieth. . . . .) item, and the screen is cleared and the score printed.

The next line is the new way of deciding whether to continue the game or stop. The question DO YOU WANT TO CONTINUE? is asked, and instructions are given for answering. An answer of this type (Y or N) has to be followed by hitting ENTER; later on we'll look at methods of answering questions like this without using the ENTER key.

This looks like a good time to sit back and take a hard look at our program, which is possible only if you have a copy on paper. Short programs of sixteen lines or less can be viewed on the video screen, but this one will not quite fit into sixteen lines. For programs longer than this, the only effective way to check it is to print it on paper or to copy the listing from the video screen.

Shape Up and Look Professional

A long hard look at our program as it is now shows that it needs renumbering. The odd-numbered lines we added have inserted useful features into the program, but they make it look rather untidy. If we had a really long program here, the simplest way of renumbering would be to use a renumbering program. As we're remodeling the program with this new random selection feature, we might as well write out the program again and renumber as we go. The result is shown in Program Listing 7.

It's at this stage that you can make a program look and run more professionally than most home-brewed efforts. One pointer is neat printing, with good tabulation and even lines, preferably right justified. Right justified means that the ends of the lines on the right of the screen are lined up, and it has to be done by careful attention to the spaces between the words in the line. A professional programmer may spend as much time tidying up the printing in a program as on the rest of the program.

The next item on the list is error traps. Professional programmers write programs which other people are going to use, and a good program should be user friendly and crash-proof. User friendly means that when the user has

to make some sort of choice, the questions should be politely put and easy to answer.

For example, it's a whole lot friendlier to be asked to type YES or NO than 1 or 2. Crash-proofing is even more important and means that every input from the user has to be tested. For example, if a YES or NO answer is called for, what happens if the user types YO or NES? A home-brew program might terminate, or worse still, it might take the answer as being YES or NO with no indication to the user. A much better way is to respond to a wrong answer with a statement such as:

```
"I'M SORRY—I DON'T RECOGNIZE THAT ANSWER ";
N$;" PLEASE TYPE YES OR NO"
```

In this line, N$ would be the word which the user had typed and the line would be followed by a GOTO instruction so that the choice was presented again.

Each request should be accompanied by a clear list of what the choices are, the user should be reminded of the choice, once made, and an unacceptable answer should be explained, with a return to the request. Making sure that this is all done is not so simple; it can take up a lot of time and needs a lot of careful thought. It also needs memory space. It pays off handsomely in the end, however, because your program will always be a delight to run, easy for you or your friends to use, and a very attractive item if you want to sell it.

Speed It Up

A few final details will help the program to run faster. It's not giving secrets away to tell you that the TRS-80 can store numbers in three different forms. If you don't specify what you want, all number variables are stored as single precision numbers, as if they consisted of a number with several places of decimals. This takes up a lot more memory space than a simple whole number (an integer). If we can define all number variables as integers, our programs will run faster and use less memory. The program in Program Listing 7 uses a lot of number variables which could be defined as integers: A,S,T,Y,N,Z. By redefining them, we can clear enough string space for more data words. Alter line 10 to read:

```
10 CLEAR 100: DEFINT A,S,T,Y,N,Z,J:A = 0:S = 0:J = 0:CLS
```

Notice that A and A$ are entirely different variables: one is a number variable which we've now defined as an integer; the other is a string variable which is an answer to a question.

How about taking the plunge for yourself and designing your own question and answer game? Remember that you will have to insert a larger number after CLEAR in line 10 if you use a lot of word pairs (the number should be equal to the number of characters, plus a bit in reserve). You will also have to change the title and instructions to fit your own ideas.

Sign	Meaning
=	exactly equal to
A<B	A less than B (earlier in the alphabet)
A>B	A more than B (later in alphabet)
A<>B or A><B	A not equal to B
A< = B	A less than or equal to B
A> = B	A greater than or equal to B

Table 1

"Into the 80s" will continue in Volume II of the Encyclopedia.

Please note: the following listings are not formatted. Enter them normally.

Program Listing 1

```
10 CLS
20 PRINT@26,"COLLECTIVES"
30 PRINT:PRINT"I SHALL GIVE YOU THE NAME OF A CREATURE.
   "
40 PRINT"I SHALL THEN ASK YOU THE NAME FOR A GROUP OF S
   UCH CREATURES"
50 PRINT"FOR EXAMPLE - WOLF"
60 PRINT "YOUR REPLY SHOULD BE - PACK.  NOW TRY ---"
70 PRINT "LION"
80 INPUT N$
90 IF N$="PRIDE" THEN PRINT "WELL DONE":GOTO120
100 PRINT "WRONG, I'M AFRAID - TRY AGAIN"
110 GOTO 70
120 END
```

Program Listing 2

```
90 IF N$="PRIDE" THEN PRINT "WELL DONE":END:ELSE PRINT
   "WRONG - I'M AFRAID, TRY AGAIN":GOTO70
```

Program Listing 3

```
10 CLS
20 PRINT@26,"COLLECTIVES"
30 PRINT:PRINT"I SHALL GIVE YOU THE NAME OF A CREATURE"
40 PRINT"I SHALL THEN ASK YOU THE NAME FOR A GROUP OF S
   UCH CREATURES"
50 PRINT"FOR EXAMPLE - WOLF"
60 PRINT"YOUR REPLY SHOULD BE - PACK.  NOW TRY------"
70 READ Q$,A$
80 PRINT Q$
90 INPUT N$
100 IF N$=A$ THEN PRINT "WELL DONE":GOTO 70:ELSE PRINT
   "WRONG, I'M AFRAID - TRY AGAIN":GOTO80
110 DATA "LION","PRIDE","WHALE","SCHOOL","FISH","SHOAL"
   ,"SHEEP","FLOCK","COWS","HERD","GEESE","GAGGLE"
```

Program Listing 4

```
10 CLS:A=0:S=0
20 PRINT@26,"COLLECTIVES"
30 PRINT:PRINT"I SHALL GIVE YOU THE NAME OF A CREATURE"
40 PRINT"I SHALL THEN ASK YOU THE NAME FOR A GROUP OF S
   UCH CREATURES"
50 PRINT"FOR EXAMPLE - WOLF":PRINT"YOUR REPLY SHOULD BE
   - PACK"
60 PRINT"YOU ARE ALLOWED THREE TRIES. AFTER THE THIRD I
   NCORRECT ANSWER":PRINT"YOU WILL BE SHOWN THE CORRE
   CT ANSWER AND ASKED THE NEXT QUESTION"
70 READ Q$,A$:T=1
75 IF Q$="Z" THEN 120
80 PRINT Q$:IF T=4 THEN PRINT "ANSWER IS ";A$;:PRINT:GO
   TO70
90 INPUT N$:A=A+1
100 IF N$=A$ THEN PRINT "WELL DONE":S=S+1:GOTO70:ELSE P
   RINT "WRONG, I'M AFRAID - TRY AGAIN":T=T+1:GOTO80
```

```
110 DATA "LION","PRIDE","WHALE","SCHOOL","FISH","SHOAL"
    ,"SHEEP","FLOCK","COWS","HERD","GEESE","GAGGLE","Z
    ","Z"
120 PRINT:PRINT"YOUR SCORE IS ";S;" IN ";A;" ATTEMPTS":
    END
```

Program Listing 5

```
10 FOR N=1TO500:PRINT "JUST LOOK AT THIS...!":NEXT
```

Program Listing 6

```
10 CLS:A=0:S=0
20 PRINT@26,"COLLECTIVES"
30 PRINT:PRINT"I SHALL GIVE YOU THE NAME OF A CREATURE"
40 PRINT"I SHALL THEN ASK YOU THE NAME FOR A GROUP OF S
     UCH CREATURES"
50 PRINT"FOR EXAMPLE - WOLF":PRINT"YOUR REPLY SHOULD BE
     - PACK"
60 PRINT"YOU ARE ALLOWED THREE TRIES. AFTER THE THIRD I
     NCORRECT ANSWER":PRINT"YOU WILL BE SHOWN THE CORRE
     CT ANSWER AND ASKED THE NEXT QUESTION"
65 T=1:Y=RND(6):FOR N=1TOY
70 READ Q$,A$:NEXT:RESTORE
75 IF Q$="Z" THEN 120
80 PRINT Q$:IF T=4 THEN PRINT "ANSWER IS ";A$;:PRINT:GO
     TO65
90 INPUT N$:A=A+1
100 IF N$=A$ THEN PRINT "WELL DONE":S=S+1:GOTO65:ELSE P
     RINT "WRONG, I'M AFRAID - TRY AGAIN":T=T+1:GOTO80
110 DATA "LION","PRIDE","WHALE","SCHOOL","FISH","SHOAL"
     ,"SHEEP","FLOCK","COWS","HERD","GEESE","GAGGLE","Z
     ","Z"
120 PRINT:PRINT"YOUR SCORE IS ";S;" IN ";A;" ATTEMPTS":
     END
```

Program Listing 7

```
10 CLS:A=0:S=0:J=0
20 PRINT@26,"COLLECTIVES"
30 PRINT:PRINT"I SHALL GIVE YOU THE NAME OF A CREATURE"
40 PRINT"I SHALL THEN ASK YOU THE NAME FOR A GROUP OF S
     UCH CREATURES"
50 PRINT"FOR EXAMPLE - WOLF":PRINT"YOUR REPLY SHOULD BE
     - PACK"
60 PRINT"YOU ARE ALLOWED THREE TRIES. AFTER THE THIRD I
     NCORRECT ANSWER":PRINT"YOU WILL BE SHOWN THE CORRE
     CT ANSWER AND ASKED THE NEXT QUESTION"
70 T=1:Y=RND(6):FORN=1TOY
80 READ Q$,A$:NEXT:RESTORE
90 PRINT Q$:J=J+1:IF T=4 THEN PRINT"ANSWER IS ";A$:PRIN
     T:GOTO120
100 INPUT N$:A=A+1
110 IF N$=A$ THEN PRINT "WELL DONE":S=S+1:GOTO120:ELSE
     PRINT "WRONG, I'M AFRAID - TRY AGAIN":T=T+1:GOTO90
120 FOR Z=1TO500:NEXT:IF J/5=INT(J/5) THEN CLS:PRINT "Y
     OUR SCORE IS ";S;" IN ";A;" ATTEMPTS":ELSE 70
130 PRINT:PRINT "DO YOU WANT TO CONTINUE? TYPE Y FOR YE
     S, N FOR NO"
140 INPUT Z$:IF Z$="Y" THEN 70 ELSE IF Z$="N" THEN END
     ELSE 140
150 DATA "LION","PRIDE","WHALE","SCHOOL","FISH","SHOAL"
     ,"SHEEP","FLOCK","COW","HERD","GEESE","GAGGLE"
```

UTILITY

Printer Calibration

Delay Loop

Printer Calibration

by L. O. Rexrode

If you use more than one type of paper with your printer, this simple modification makes a lot of sense. At the office my TRS-80 and Centronics 779 with tractor feed are primarily used for handling advertising mail inquiries and our normal operation uses five different types of forms.

In the interest of saving time and improving efficiency, the need for a method that starts every print routine exactly where it is intended, regardless of who is operating the sytem, was apparent.

Three Variables

With the Centronics 779, three variables are of primary importance in printing a professional looking form: left margin setting, top-of-form setting, and the density setting. Any, or all of these, may require different settings for various print routines or form sizes. The length of form (number of lines) is not included, since a program is usually written with this factor as a constant. The simplest variable is the left margin setting. The printer's roller/tear-bar assembly has an engraved scale in one-eighth-inch increments from – 15 on the left to 90 on the right. This scale should be used to set the left edge of the form to be printed. The value is determined by a trial run.

To set the top-of-form requires the addition of a scale to the 779. This is done quite easily if you use the holddown clamp on the left pin feed assembly. Use a three-and-one-half-inch peel-off label and type a column of numbers, say, from 1 to 21, trim off the excess margins, and place the label right along the edge of the holddown clamp. The resultant scale now allows you to set the top-of-form to a specified position.

The third variable is the print density control. This sets the number of characters-per-inch printed. For maximum legibility you usually want this set as wide as the form will allow. The control, located on the rear of the printer, is not only difficult to reach but requires several lines of print, with trial and error adjustments, to get the optimum settings. A calibrated dial knob solves this problem easily.

The dial knob I used is Radio Shack's part no. 274-413. However, on some printers, the shaft may not protrude quite far enough to get a good bite when the set screw is tightened. If so, use a coarse file on the rear face of the knob, removing about 1/32" to 1/16".

Cut a sliver from the adhesive-backed label to make a pointer for the dial and place it approximately as shown in Figure 1. Rotate the adjustment shaft fully counter-clockwise (lowest density setting) and push the knob onto

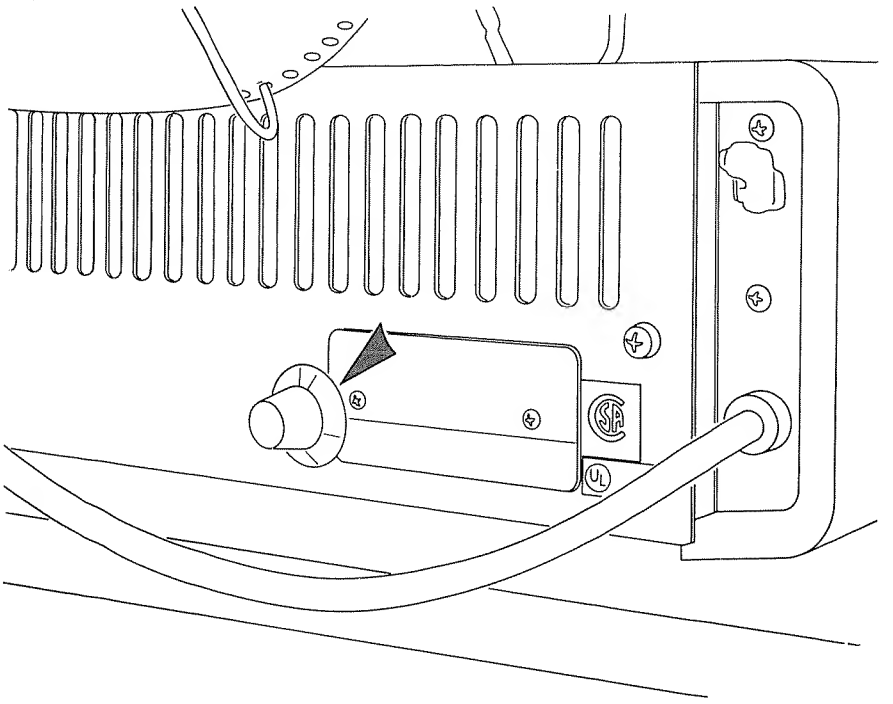
the shaft, setting the number one mark to the arrow. Tighten the set screw and you're through.

Find Your Values

Run each of your print routines and determine the optimum value of the three variables: left margin, top-of-form, and density. Note these values for use with the routines. The best place to note them is in your program, immediately preceding the first LPRINT statement. If you are new to programming, here is the statement I use (your line number, title, and values will relate to your own program, of course):

```
CLS:PRINT@320, "READY TO PRINT MAILING LABELS DIRECTLY FROM
DISK":PRINT:PRINT"SET TOP-OF-FORM TO 20":PRINT"SET LEFT EDGE OF
FORM TO -4":PRINT "SET DENSITY CONTROL TO 4": INPUT" 'ENTER' TO
CONTINUE";X
```

When LLISTing a program in BASIC, it is not possible to use such a prompting message. I am constantly LLISTing with the density set too low—which results in my losing the ends of long program statements. As a reminder to myself to set the density before starting a LLIST, I used another label, placed prominently on the top front of the printer that almost shouts,



"For Listings Set the Density to 7." It works. We have been using this method for several months now, and it is quite gratifying to see the results. No matter who is operating the system, and regardless of how often we change forms or paper, the first line of print goes down exactly where we want it to.

A couple of other time-savers we have added to our program are worth the time and the slight additional memory required. The first one resulted from our operator having spent over an hour trying to figure out why the computer wouldn't run and finally realizing that the out-of-paper switch was off, due to a small tear in the form which wasn't readily apparent.

To prevent this from happening again, I added a GOSUB in front of every LPRINT statement. The subroutine checks the status of both the out-of-paper switch and the print switch. If either is off, a prompting message is displayed telling the operator what is wrong and allowing him to correct before continuing. My subroutine reads as follows:

```
750 IFPEEK(14312)<128 RETURN
751 CLS:PRINT@468,"PRINTER NOT READY":
    PRINT@ 583, "'R'—RETURN
    TO MENU 'O'—OK TO CONTINUE"
752 Q$ = INKEY$ :IF Q$ = "R"THEN 511
753 IFQ$ = "O"THEN RETURN ELSE 752
```

This simple solution has saved us hours.

Add "Top-of-Form"

Since the 779 doesn't have front panel controls that allow either a line feed or a top-of-form, I added these as a part of my program. Since each program has a menu, this was the obvious place to access these routines. The menu uses the INKEY$ function, so it was just a matter of adding the up arrow as the selection to advance one line and adding the letter T to go to top-of-form. I did not include these two symbols in the menu table to prevent clutter. They are blind selections. Here are the routines:

```
620 IFQ$ = "↑"THEN 800
625 IFQ$ = "T"THEN 810

800 IF PEEK(14312)<128 GOTO 805 ELSE 511
805 POKE 16424,1;POKE 16425,0:LPRINT CHR$(11): GOTO 511
810 IF PEEK(14312)<128 GOTO815 ELSE 511
815 POKE 16424,51:POKE 16425,0;LPRINT CHR$(11):GOTO511
```

In either of these routines, if the printer is not ready, nothing happens except returning you to the menu. The ↑ advances the paper one line. The T advances the paper 51 lines, then you return to the menu.

In all the routines in this article, my menu routine begins at line 511.

Delay Loop

by Allan S. Joffe W3KBM

If you've ventured into the world of assembly programming, you'll eventually need a delay or timing loop. Though there is no "best way" to delay a program, we can explore various methods.

Basically a time delay loads a value into a register (or values into a pair of registers) and then decrements this value to zero. This means we also need a mechanism that tells the computer when it has decremented the initial value to zero, so that it can continue the program.

The Zero Flag

Consider Program Listing 1, loaded using T-BUG.

The C2 instruction in memory location 4A06 is a JP NZ,NN which signals the computer that if the comparison of the contents of the D and A registers is not zero, it should go back to the location specified by the NN which in our case is 4A02. This is the location of the decrement instruction.

One thing that this routine does not take into account is that when decrementing a single register, the zero flag is operative. (This is not true when we come to the case of decrementing a pair of registers.)

You notice that to continue the decrementing process, we had to tell the program to go back to a specific location. This means that if you did not load the program into the memory locations specified, you would have had to change the return locations in 4A07 and 4A08.

In Program Listing 2 we will use a relative jump instruction that obviates the need for such changes. This allows the program to relocate, for with the relative jump, the computer knows where it should go even if you change the origin of the routine.

In this listing I've introduced the mnemonic form of the instruction.

This listing is considerably shorter than the first one for several reasons. One is that the first example is used to illustrate the idea of comparing the contents of one register with another to determine when the zero condition is reached. The first listing tells the program where to go if the zero condition has not been reached. This takes three locations, while the new method uses but two locations in memory.

The jump relative instruction in location 4A03 works in conjunction with the FD information in location 4A02. We know that to make the loop decrement, we have to keep returning to location 4A04. Location 4A02 is three steps back from location 4A04, counting location 4A04 as step one. You have signaled the computer that this is what you want by inserting FD.

What is the significance of FD? It is a minus three in twos complement form. If you had needed a relative jump of minus seven, then the FD would have become F9. Table 1 shows the relative jump values from minus one to minus 16.

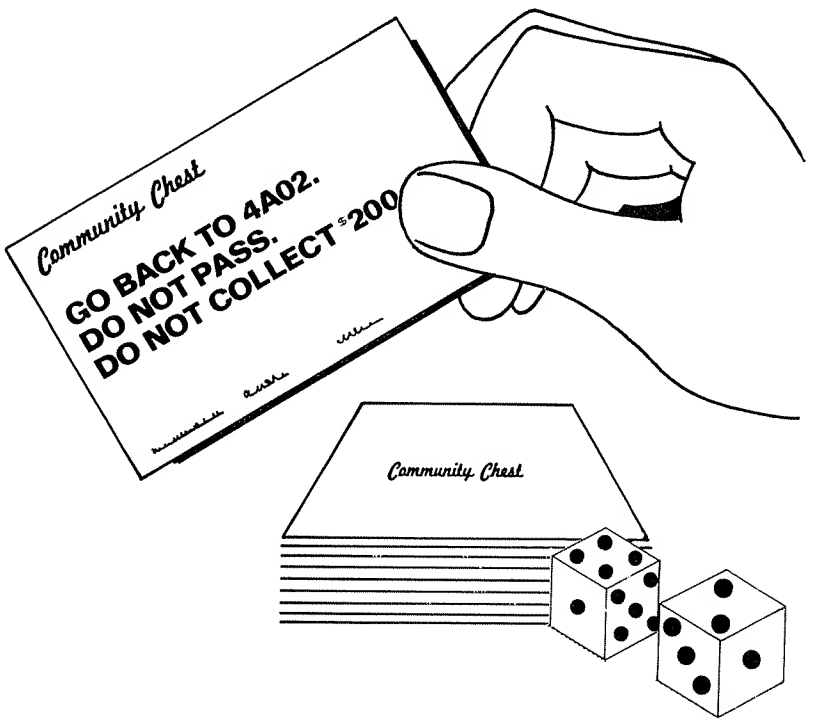
The absolute delay produced by the simple timing loop using a single register is just about maximum when the register initial value is FF (255 decimal). I say "just about" because if the initial value in the register is 00, then you have 256 iterations of the loop, because the first decrement takes you from 00 to FF.

This is a bone for the nitpickers in the group.

Longer Delays

If we need longer delays, we can insert another identical loop after the first one, but there is another route to travel. This method decrements a pair of registers. If one register can be packed with FF, then two registers can be packed with FFFF.

Remember, if we use a pair of registers we will have to resort to some sort of a compare operation as with Program Listing 1, because when you decrement a pair of registers, the zero flag is not automatically working in your



behalf. Thus we need some program steps to get the zero flag back from vacation.

Since the idea of the jump relative code seems to have merit, we will go that route as well. When possible, I like to use the BC register pair for decrementing. BC seems to shout out "byte counter" and is, for me, a memory jogger.

Running Program Listing 3, you notice that it takes more time before the program returns to T-BUG. This tells us that we have achieved a much longer delay than when decrementing the contents of a single register. The delay is in the neighborhood of 1.1 seconds.

The listing also uses the OR function, which is why we had to do what we did in memory location 4A04.

To operate any of the logic functions, you have to call on the services of the A register.

The jump relative figure in location 4A08 is equal to minus six (see Table 1), the proper value to get back to location 4A03, which contains the DEC BC instruction.

The above is but one viable routine, not the only one.

FF	−1
FE	−2
FD	−3
FC	−4
FB	−5
FA	−6
F9	−7
F8	−8
F7	−9
F6	−10
F5	−11
F4	−12
F3	−13
F2	−14
F1	−15
F0	−16

Table 1. *Minus Relative Jump Values (twos complement form)*

The Interrupt

Now let us get down to the business of combining a single register decrement and the register pair decrement to achieve significantly longer time delays. The game plan is to interrupt the decrement of the register pair while a single register decrements to zero. BC is the register pair in Program Listing 4 and the D register is used as the interrupt register.

When you load Program Listing 4 and RUN, be prepared to sit back for two minutes and 32 seconds before it returns you to T-BUG. The key ele-

ment of the time delay is the value in location 4A05, the value being decremented in the D register.

If you change this value from FF to AA, then the delay time becomes one minute and 43 seconds. If it is changed to 64, then the time delay is one minute and one second. Changing this value to OA is going to give you a total time delay of about eight seconds. Thus this delay routine is quite flexible and should satisfy all but the most unusual needs.

You will get the most good out of this exercise if you use the breakpoint of T-BUG to examine the program at various points.

```
4A00   16   Load register D with N (value to decrement)
4A01   FF   Hex for 255
4A02   15   Decrement value in D register
4A03   3E   Load the A register with zero . . . this is
4A04   00   the test value to show loop is done
4A05   BA   Compare value in D with value in A
4A06   C2   If this test shows A and D both zero the
4A07   02   loop is finished. If not pgm goes to 4A02
4A08   4A   and continues decrementing value in D
4A09   C3   When decrement is finished the program
4A0A   80   returns to T-BUG which is at 4380.
4A0B   43
```

Program Listing 1

```
4A00   16   LD D,N
4A01   FF
4A02   15   DEC D
4A03   20   JRNZ
4A04   FD
4A05   C3
4A06   80
4A07   43
```

Program Listing 2

```
4A00   01   LD BC,NN
4A01   FF
4A02   FF
4A03   0B   DEC BC
4A04   78   LD A,B
4A05   B0   OR B
4A06   B1   OR C
4A07   20   JR NZ
4A08   FA
4A09   C3
```

Program continued

```
4A0A  80
4A0B  43
```

Program Listing 3. *Decrementing the BC register pair*

```
4A00  01  LD BC,NN
4A01  FF
4A02  FF
4A03  0B  DEC BC
4A04  16  LD D,N
4A05  FF
4A06  15  DEC D
4A07  20  JR NZ
4A08  FD
4A09  78  LD A,B
4A0A  B0  OR B
4A0B  B1  OR C
4A0C  20  JR NZ
4A0D  F5
4A0E  C3
4A0F  80
4A10  43
```

Program Listing 4

APPENDIX

APPENDIX A

BASIC Program Listings

Debugging someone else's mistakes is no fun. In a business environment, where programs are continuously updated and programmers come and go, well-commented and structured programs are a must. Indeed, it behooves any serious programmer to learn structured technique.

The BASIC language has no inherent structure. Most interpreters allow remark lines and some are capable of ignoring unnecessary spacing, but BASIC is still more "Beginner's Instruction Code" than "All-purpose."

The listings in this encyclopedia are an attempt at formatting the TRS-80 BASICs. We think it makes them easier to read, easier to trace, and less imposing when it comes time to type them into the computer. You should *not*, however, type them in exactly as they appear. Follow normal syntax and entry procedures as described in your user's manual.

Level I Programs

Programs originally in Level I have been converted to allow running in Level II. To run in Level I follow this procedure.

- Delete any dimension statements. Example: DIM A (25).
- Change PRINT@ to PRINTAT.
- Make sure that no INPUT variable is a STRING variable.
  Example: INPUT A$ would be changed to INPUT A and subsequent code made to agree.
- Abbreviate all BASIC statements as allowed by Level I.
  Example: *PRINT* is abbreviated *P.*

Model III Users

For the Model I, OUT255,0 and OUT255,4 turn the cassette motor off and on, respectively. For the Model III, change these statements to OUT236,0 and OUT236,2.

APPENDIX B

Glossary

A

access time—the elapsed time between a request for data and the appearance of valid data on the output pins of a memory chip. Usually 200-450 nanoseconds for TRS-80 RAM.

accumulator—traditionally the register where arithmetic (the accumulation of numbers) takes place.

acoustic coupler—a connection to a modem allowing signals to be transmitted through a regular telephone handset.

A/D converter—analog to digital converter. See D/A converter.

address—a code that specifies a register, memory location, or other data source or destination.

ALGOL—an acronym for ALGOrithmic Language. A very high-level language used in scientific applications.

algorithm—a predetermined process for the solution of a problem or completion of a task in a finite number of steps.

alignment—adjustment of hardware to achieve proper transfer of data. In the TRS-80 this usually applies to cassette heads and disk drives.

alphanumerics—refers to the letters of the alphabet and digits of the number system, specifically omitting the characters of punctuation and syntax.

ALU—Arithmetic-Logic Unit. Internal, and inaccessible to the programmer, it is the interface between registers and memory, manipulating them as necessary to perform the individual instructions of the microprocessor.

analog—data is represented electrically by varying voltages or amplitudes.

AND—a Boolean logical function. Two operators are tested and if both are true the answer is true. Truth is indicated by a high bit, or "1" in machine language, or a positive value in BASIC. If the operators are bytes or words, each element is tested separately.

APL—a popular high-level mathematical language.

argument—any of the independent variables accompanying a command.

ASCII—American Standard Code for Information Interchange. An almost universally accepted code (at least for punctuation and capital letters) where characters and printer commands are represented by numbers between 0 and 255 (base 10). The number is referred to as an ASCII code.

assembler—a piece of software that translates operational codes into their binary equivalents.

B

backup—refers to making copies of all software and data stored externally and having duplicate hardware available.

base—a mathematical term that refers to the number of digits in a number system. The decimal system, using digits 0 through 9, is called base 10. The binary system is base 2.

BASIC—an acronym for Beginner's All-purpose Symbolic Instruction Code. Developed at Dartmouth College and similar to FORTRAN. The standard, high-level, interactive language for microcomputers.

batch processing—a method of computing where many of the same type jobs or programs are done in one machine run. For example, a programming class may type programs on data cards and turn them over to the computer operator. All the cards are put into the card reader, and the results of each person's program are returned later. This is contrasted with interactive computing.

baud rate—a measure of the speed at which serial data is sent. The equivalent of bits per second (bps) in microcomputing.

benchmark—to test performance against a known standard.

binary—a number system which uses only 0 and 1 as digits. It is the equivalent of base 2. Used in microcomputing because it is easy to represent 1s and 0s by high and low electrical signals.

bit—an abbreviation for binary digit. A 0 or 1 in the binary number system. A single high or low signal in a computer.

Boolean algebra—a mathematical system of logic named after George

Boole. Routines are described by combinations of ANDs, ORs, XORs, NOTs, and IF-THENs. All computer functions are based upon these operators.

boot—short for bootstrap loader or the use of one. The bootstrap loader is a very short routine whose purpose is to load a more sophisticated system into the computer when it is first turned on. Sometimes it is keyed in, and on other machines it is in read only memory (ROM). Using this program is called "booting" the system or cold-starting.

bps—bits per second.

buffer—memory set aside temporarily for use by the program. Particularly refers to memory used to make up differences in the data transfer rates of the computer and external devices such as printers and disks.

bug—an error in software.

bus—an ordered collection of all address, data, timing, and status lines in the computer.

byte—eight bits that are read simultaneously as a single code.

C

CAI—an acronym for Computer Aided Instruction.

card—a specifically designed sheet of cardboard with holes punched in specific columns. The placement of the holes represents machine-readable data. Also a term referring to a printed circuit board.

carrier—a steady signal that can continuously be slightly modified (modulated). These modulations can be interpreted as data. In microcomputers the technique is especially used in modem communications and tape input/output (I/O).

character—a single symbol that is represented inside the computer by a specific code.

checksum—a method of detecting errors in a block of data by adding each piece of data in the block to a sum and comparing the final result to a predetermined result for that block of data.

chip—a physical package containing electrical circuits. They vary from aspirin-size for a simple timer to about the size of a stick of gum for a complete microprocessor.

clock—a simple circuit that generates the synchronization signals for the microprocessor. The speed of frequency of this clock directly affects the speed at which the computer can perform, regardless of the speed of which the individual chips are capable.

COBOL—COmmon Business Oriented Language. A language used primarily for data processing. Allows programming statements that are very similar to English sentences.

compiler—a piece of software that will convert a program written in a high-level language to binary code.

complement—a mathematical calculation. In computers it specifically refers to inverting a binary number. Any 1 is replaced by a 0, and vice versa.

concatenate—to put two things, each complete by itself, together to make a larger complete thing. In computers this refers to strings of characters or programs.

constant—a value that doesn't change.

CPU—Central Processing Unit. The circuitry that actually performs the functions of the instruction set.

CRT—Cathode Ray Tube. In computing this is just the screen the data appears on. A TV has a CRT.

cue—refers to positioning the tape on a cassette unit so that it is set up to read/write the right section of tape.

cycle—a specific period of time, marked in the computer by the clock.

D

D/A converter—digital to analog converter. Common in interfacing computers to the outside world.

data base—refers to a series of programs each having a different function

but all using the same data. The data is stored in one location or file and each program uses it in a fashion that still allows the other program to use it.

data entry—the practice of entering data into the computer, or onto a storage device. Knowledge of operating or programming a computer is not necessary for a data entry operator.

debug—to remove errors from a program.

decrement—to decrease the value of a number. In computers the number is in memory or a register, and the amount it is decremented is usually one.

dedicated—in computer terminology, a system set up to perform a single task.

default—that which is assumed if no specific information is given.

degauss—to demagnetize. Must be done periodically to tape and disk heads for reliable data transfer.

digital—all data is represented in binary code. In microcomputers, a high electrical signal is a 1 and a low signal is a 0.

disassembly—remaking an assembly source program from a machine-code program.

disk—an oxide-coated, circular, flat object, in a variety of sizes and containers, on which computer data can be stored.

disk controller—an interface between the computer and the disk drive.

disk drive—a piece of hardware that rotates the disk and performs data transfer to and from the disk.

disk operating system (DOS)— the system software that manipulates the data to be sent to the disk controller.

DMA—direct memory access. A process where the CPU is disabled or bypassed temporarily and memory is read or written to directly.

documentation—a collection of written instructions necessary to use a piece of hardware, software, or a system.

dot matrix printer—instead of each letter having a separate type head (like a typewriter), the single print head makes the characters by printing groups of dots. The print is not as easy to read, but such printers are less expensive to make.

driver—a small piece of system software used to control an external device such as a keyboard or printer.

dump—to write data from memory to an external storage device.

duplex—refers to two-way communications taking place independently, but simultaneously.

dynamic memory—circuits that require a periodic (every few milliseconds) recharge so that the stored data is not lost.

E

EAROM—an acronym for Electrically Alterable Read Only Memory. The chip can be read at normal speed, but must be written to with a slower process. Once written to, it is used like a ROM, but can be completely erased if necessary.

editor—a program that allows text to be entered into memory. Interactive languages usually have their own editor.

EOF—End Of File.

EOL—End Of Line (of text).

EPROM—Electrically Programmable Read Only Memory. The chip is programmed by voltages higher than normal for computer chips. Once programmed, it is used like ROM, but can be erased by exposure to ultraviolet light.

exclusive OR—see XOR.

execution cycle—a cycle during which a single instruction actually occurs.

expansion interface—a device attached to the computer that allows a greater amount of memory or attachment of other peripherals.

F

fetch cycle—a cycle during which the next instruction to be performed is read from memory.

file—a set of data, specifically arranged, that is treated as a single entity by the software or storage device.

firmware—software that is made semi-permanent by putting it into some type of ROM.

flag—a single bit that is high (set) or low (reset), used to indicate whether or not certain conditions exist or have occurred.

flowcharting—a method of graphically displaying program steps, used to develop and define an alogrithm before writing the actual code.

FORTRAN—FORmula TRANslator. One of the first high-level languages, written specifically to allow easy entry of mathematical problems.

G

game theory—see von Neumann.

garbage—computer term for useless data.

gate—a circuit that performs a single Boolean function.

GIGO—Garbage In, Garbage Out. One of the rules of computing. If the data going into the computer is bad, the data coming out will be bad also.

graphics—information displayed pictorially as opposed to alphanumerically.

H

half duplex—data can flow in both directions, but not simultaneously. See duplex.

handshaking—a term used in data transfer. Indicates that beside the data lines there are also signal lines so both devices know precisely when to send or receive data. Contrast with buffer.

hangup—a situation where it seems the computer is not listening to you.

hard copy—a printout.

hardware—refers to any physical piece of equipment in a computer system.

hexadecimal—a number system based on sixteen. The decimal digits 0-9 are used along with the alpha characters A-F, which are also recognized as digits.

high—a signal line logic level. The computer senses this level and treats it as a binary 1.

high-level language—a programming code that does not require a knowledge of the CPU structure.

high order—see most significant.

HIT—acronym for Hash Index Table. A section of the directory on a TRS-80 disk.

host computer—the primary computer in a multi-computer or terminal hookup.

human engineering—usually refers to designing hardware and software with ease of use in mind.

I

IC—integrated circuit. See chip.

immediate addressing—the address of the information that an operation is supposed to act upon immediately follows the operation code.

increment—to increase, usually by one. See decrement.

indexed—the information is addressed by a specified value, or by the value in a specified register.

indirect—the address given points to another address, and the second address is where the information actually is.

intelligent terminal—a terminal with a CPU and a certain amount of

memory that can organize the data it receives and thus achieve a high level of handshaking with the host computer.

interactive computing—refers to the appearance of a one-to-one human-computer relationship.

interface—a piece of hardware, specifically designed to hook two other devices together. Usually some software is also required.

interpreter—a piece of system software that executes a program written in a high-level language directly. While useful for interactive computing, this system is too slow for most serious programming. Contrast with compile.

interrupt—a signal that tells the CPU that a task must be done immediately. The registers are pushed to the stack, and a routine for the interrupt is branched to. When finished, the registers are popped from the stack and the main program continues.

I/O—acronym for input/output. Refers to the transfer of data.

J

jack—a socket where wires are connected.

K

K—abbreviation for kilo. In computer terms 1024, in loose terms 1000.

L

least significant—refers to the lowest position digit of a number, or right-most bit of a byte. In 19963 the 3 is the least significant digit. Opposite of most significant.

LIFO—acronym for Last In First Out. Most CPUs maintain a "stack" of memory that this rule applies to. The last piece of data pushed into the stack is the first piece popped out.

light pen—a device that senses light, interfaced to the computer for the purpose of drawing on the CRT screen.

loop—a set of instructions that executes itself continuously. If the programmer had the presence of mind to provide for a test, the loop is discontinued when the test is met, otherwise it goes on until the machine is shut down.

loop counter—one way to test a loop. The counter is incremented at each pass through the loop. When it reaches a certain value the loop is terminated.

low—a logic signal voltage. The computer senses this as a binary 0.

LSI—acronym for Large Scale Integration. An integrated circuit with a large number of circuits such as a CPU. See chip.

M

machine code—refers to programming instructions that are stored in binary and can be executed directly by the CPU without any compilation, interpretation, or assembly.

machine language—the primary instructions that were designed into the CPU by the manufacturer. These instructions move data between memory and registers, perform simple adding in registers, and allow branching based on values in registers.

macro—a routine that can be separately programmed, given a name, and executed from another program. The macro can perform functions on variables in the program that called it without disturbing anything else and then return control to the calling program.

mainframe—refers to the CPU of a computer. This term is usually confined to larger computers.

memory—the hardware that stores data for use by the CPU. Each piece of data (bit) is represented by some type of electrical charge. Memory can be anything from tiny magnetic doughnuts to bubbles in a fluid. Most microcomputers have chips that contain many microscopic capacitors, each capable of storing a tiny electrical charge.

microprocessor—a CPU on a single chip.

mnemonic—a short, alphanumeric abbreviation used to represent a machine language code. An assembler will take a program written in these mnemonics and convert it to machine code.

modem—MOdulator/DEModulator. An I/O device that allows communication over telephone lines.

monitor—(1) a CRT. (2) a short program that displays the contents of registers and memory locations and allows them to be changed. Monitors can also allow another program to execute one instruction at a time, saving programs and disassembling them.

most significant—refers to the highest value position of a number of the leftmost bit of a byte. In the number 1923 the 1 is most significant because it represents thousands.

N

NAND—an acronym for NOT AND. A Boolean logic expression. AND is performed, then NOT is performed to the result.

nanosecond—one billionth of a second.

nesting—putting one loop inside another. Some computers have a limit to the number of loops that can be nested.

NOT—a Boolean operator that reverses outputs (1 becomes 0, 0 becomes 1).

O

object code—all of the machine code that is generated by a compiler or assembler. Once object code is loaded into memory it is called machine code.

octal—refers to the base 8 number system, using digits 0-7.

OEM—Original Equipment Manufacturer.

offset value—a value that can be added to an address. Most addressing modes allow an offset value.

off-the-shelf—a term referring to software. It means that the program is generalized so that it can be used by a greater number of computer owners, thus it can be mass produced and bought "off-the-shelf."

on-line—a term describing a situation where one computer is connected to another, with full handshake, over a modem line.

OR—a Boolean logic function. If at least one of the lines tested is high (binary 1), the answer is high.

overflow—a situation that occurs when an arithmetic function requires more than the machine is capable of handling. Most computers have a flag so that this condition can be tested.

overlay—a method of decreasing the amount of memory a program uses by allowing sections that are not in use simultaneously to load into the same area of memory. The new routine destroys the first routine, but it can always be loaded again if needed. Usually used in system programs.

oxide—an iron compound coating on tapes and disks that allows them to be magnetized so that they can be read by electrical devices and the information converted back to machine code.

P

page—refers to a 256- (2 to the 8th power) word block of memory. How large a word is depends on the computer. Most micros are 8-bit word machines. The term is important because many chips do special indexed and offset addressing on the page where the program counter is pointing and/or on the first page of memory.

parallel—describes a method of data transfer where each bit of a word has its own data line, and all are transferred simultaneously.

parameter—a variable or constant that can be defined by the user and usually has a default value.

parity—a method of checking accuracy. The parity is found by adding all the bits of a word together. If the answer is even the parity is 0 or even. If odd, the parity is 1 or odd. The bit sometimes replaces the most significant bit and usually sets a flag.

PC board—stands for Printed Circuit board. A piece of plastic board with lines of a conductive material deposited on it to connect the components. The lines act like wires. These can be manufactured quickly and are easy to assemble the components on.

peripheral—any piece of hardware that is not a basic part of the computer.

PILOT—a simple language for handling English sentences and strings of alphanumeric characters.

PL/1—a programming language used by very large computers. It incorporates most of the better features from other programming languages.

plotter—a device that can draw graphs and curves controlled by the computer through an interface.

port—a single addressable channel used for communications.

PROM—Programmable Read Only Memory. A memory device that is written to once, and from then on acts like a ROM.

pseudo code—a mnemonic used by assemblers that is not a command to the CPU, but a command to the assembler itself.

R

RAM—an acronym for Random Access Memory. Memory that can be written to or read from. It is addressed by the address bus.

real time clock—a clock in the sense that we normally think of one, interfaced to the computer.

record—a file is divided into records, each of which is organized in the same manner.

register—a memory location used by the CPU and not addressed by the address bus. It cannot be used by the programmer.

relative addressing—an address that is dependent upon where the program counter is presently pointing.

ROM—an acronym for Read Only Memory. Memory that is addressed by the bus, but can only be read from. If you tell the CPU to write to it, the machine will try, but the data is not remembered.

RPG—an acronym for RePort Generator. A language for business that primarily reads data off cards and prints out reports containing that data.

RS-232—an interface that converts parallel data to serial data for communications purposes. The output is universally standard.

S

semiconductor—a compound that can be made to vary its resistance to electricity by mixing it differently. Layers of this material can be used to make circuits that do the same things tubes do, but using much less electricity. Transistors and integrated circuits are made from semiconductive material and called semiconductors.

serial—a way of sending data, one bit at a time, between two devices. The bits are rejoined into bytes by the receiving device. Contrast with parallel.

sign bit—sometimes the most significant bit is used to indicate the sign of the number it represents. 1 is negative (−) and 0 is positive (+).

simulator—a computer that is programmed to mimic the action and functions of another piece of machinery, usually for training purposes. A computer is usually employed because it is cheaper to have the computer simulate these actions than to use the real thing. Airplane and power plant trainers are excellent examples.

software—refers to the programs that can be run on a computer.

source program—the program written in a language or mnemonics that is converted to machine code. The source program as well as the object code generated from it can be saved in mass storage devices.

stack—an area of memory used by the CPU and the programmer particularly for storage of register values during interrupt routines. See LIFO.

status register—the register that contains the status flags set and tested by the CPU operations.

stepper motor—a special motor in a disk drive that moves the read/write head a specific distance each time power is applied. That distance defines the tracks on a disk.

subroutine—a routine within a program that ends with an instruction to return program flow to where it branched from to get to the routine. This routine is used many times from many different places in the program, and the subroutine allows you to only write the code for that routine once. Similar to a macro.

syntax—the term is used exactly as it is used in English composition. Every

language has its own syntax.

system software—software that the computer must have loaded and running to work properly.

T

table—an ordered collection of variables and/or values, indexed in such a way that finding a particular one can be done quickly.

text editor—see word processor.

time-sharing—refers to systems which allow several people to use the computer at the same time.

track—a concentric area on a disk where data is stored in microscopic magnetized areas.

TTL—Transistor-Transistor Logic. Means that the electrical values for logic highs and lows fall within the values necessary to run transistors. See semiconductor.

U

utility—a program designed to aid the programmer in developing other software.

V

variable—a labeled entity that can take on any value.

von Neumann, John (1903-1957)—Mathemetician. Put the concept of games, winning strategy, and different types of games into mathematical formulae. Also advanced the concept of storing the program in memory as opposed to having it on tape.

W

word—in computing it refers to a number of bits that are in a parallel for-

mat. If the CPU works with 8 bits then the word length is 8 bits. Common word sizes are 4, 8, 12, 16, and 32. Some are as large as 128 bits.

word processor—a computer system dedicated to editing text and printing it in various controllable formats. See editor.

write—to store in memory or on a mass storage device.

X

XOR—a Boolean function. Acronym for eXclusive OR. Similar to OR but answer is high (1) if and only if one line is high.

Z

zero page—refers to the first page of memory.

INDEX

INDEX COMPILED BY LAURA BARNICLE

WAYNE GREEN INC.

You can find everything you need in Software and Books from Wayne Green Inc. **Load-80**, designed as a companion to **80 Microcomputing**, is a dump of the major program listings in "80" on a monthly cassette. You get

- the best and the longest of the programs in "80" without typing those lengthy listings so you save hours of your time.
- a big money savings—the programs would cost a lot more if you had to buy each one separately.
- no more aggravation (you don't have to go back and search for the typo).

To order Load-80 call Toll Free 800-258-5473.

Instant Software™ Inc.

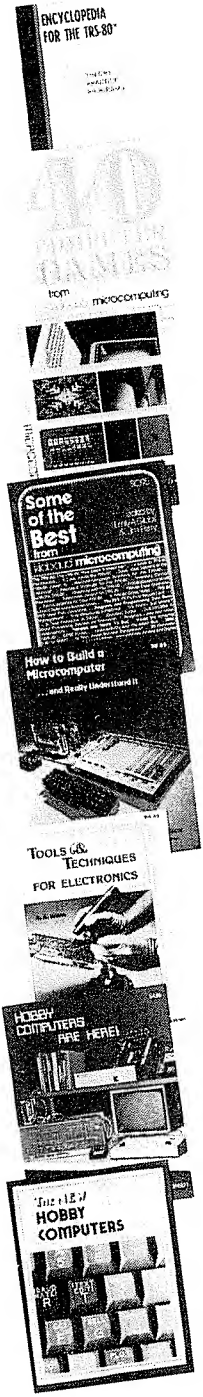
Instant Software is the most complete publisher of microcomputer software in the world. You get

- programs for every need—there are hundreds ranging from business to games to science to health to basic utilities.
- an inexpensive way to expand the function of your computer.
- programs for all the systems from Apple to TRS-80* and all the machines in between.
- the best programs available since **Instant Software** is the biggest it gets the best programmers.

For a catalog of Instant Software products call Toll Free 800-258-5473.

*TRS-80 is a trademark of the Radio Shack Division of Tandy Corp

WAYNE GREEN BOOKS



Encyclopedia for The TRS-80* —A ten-volume series to be issued every two months starting July 1981. The Encyclopedia contains the most up-to-date information on how to use your TRS-80*.

40 Computer Games from Kilobaud Microcomputing—Games in nine different categories for large and small systems, including a section on calculator games.

Understanding and Programming Microcomputers—A well-structured introductory text on the hardware and software aspects of microcomputing.

Some of the Best from Kilobaud Microcomputing—A collection of articles focusing on programming techniques and hardcore hardware construction projects.

How to Build a Microcomputer and Really Understand It—A technical manual and programming guide that takes the hobbyist step-by-step through the design, construction, testing and debugging of a complete microcomputer system (6502 chip).

Tools and Techniques for Electronics—Describes the safe and correct ways to use basic and specialized tools for electronic projects as well as specialized metal working tools and the chemical aids which are used in repair shops.

Hobby Computers Are Here—The fundamentals on how computers work—explaining the circuits and the basics of programming plus a couple of TVT construction projects.

The New Hobby Computers—Contains introductory articles on such subjects as large scale integration, how to choose a microprocessor chip, and introduction to programming, computer arithmetic, etc.

To order call Toll Free 800-258-5473.

*TRS-80 is a trademark of Radio Shack Division of Tandy Corp.

The real value of your computer lies in your ability to use it. The capabilities of the TRS-80* are incredible if you have the information which will help you get the most from it. Little of this information is available in your instruction books.

The *Encyclopedia for the TRS-80* will teach you how to get the most from your computer. In addition to a wealth of programs which are ready for you to use, reviews of accessories and commercially available programs, you will also learn how to write your own programs or even modify commercial programs for your own specific use.

The *Encyclopedia* is packed with practical information, written and edited for the average TRS-80 owner, not the computer scientist. You will find it interesting and valuable.

Wayne Green
Publisher

ISBN 0-88006-026-3